

OTRS 3.0 - Developer Manual

OTRS 3.0 - Developer Manual

Copyright © 2003-2010 OTRS AG

René Bakker, Hauke Böttcher, Stefan Bedorf, Shawn Beasley, Jens Bothe, Udo Bretz, Martin Edenhofer, Martin Gruner, Manuel Hecht, Christopher Kuhn, André Mindermann, Henning Oschwald, Thomas Raith, Stefan Rother, Burchard Steinbild

This work is copyrighted by OTRS AG.

You may copy it in whole or in part as long as the copies retain this copyright statement.

The source code of this document can be found at source.otrs.org [<http://source.otrs.org/viewvc.cgi/doc-developer/>].

UNIX is a registered trademark of X/Open Company Limited. Linux is a registered trademark of Linus Torvalds.

MS-DOS, Windows, Windows 95, Windows 98, Windows NT, Windows 2000, Windows XP, Windows 2003 and Windows Vista are registered trademarks of Microsoft Corporation. Other trademarks and registered trademarks are: SUSE and YaST of SUSE Linux GmbH, Red Hat and Fedora are registered trademarks of Red Hat, Inc. Mandrake is a registered trademark of MandrakeSoft, SA. Debian is a registered trademark of Software in the Public Interest, Inc. MySQL and the MySQL Logo are registered trademarks of MySQL AB.

All trade names are used without the guarantee for their free use and are possibly registered trade marks.

OTRS AG essentially follows the notations of the manufacturers. Other products mentioned in this manual may be trademarks of the respective manufacturer.

Table of Contents

1. Getting Started	1
Development Environment	1
Framework checkout (CVS)	1
Linking Expansion Modules	1
Necessary Actions after Linking	2
Architecture Overview	2
Directories	2
Files	2
Core Modules	3
Frontend Handle	3
Frontend Modules	3
CMD Frontend	3
Database	4
2. Hacking OTRS	5
How it Works	5
Config Mechanism	5
Database Mechanism	9
Log Mechanism	13
Skins	14
The CSS and JavaScript "Loader"	18
Templating Mechanism	24
How to Extend it	32
Module Format	32
Object Basics	105
Writing an OTRS module for a new object	108
How to Publish your OTRS Extensions	113
Package Management	113
Package Building	114
How to Upgrade your OTRS Extensions to Newer Versions of OTRS	122
Upgrading OTRS Extensions from 2.4 to 3.0	122
3. Contributing to OTRS	127
Translating OTRS	127
How it works	127
Updating an existing translation	129
Adding a new frontend translation	129
Translating the Documentation	130
Code Style Guide	131
Perl	131
JavaScript	135
CSS	136
User Interface Design	137
Capitalization	137
Accessibility Guide	138
Accessibility Basics	138
Accessibility Standards	139
Implementation guidelines	139
Unit Tests	141
Creating a test file	141
Testing	142
True()	143
False()	143

Is()	143
A. Additional Resources	144
OTRS.org	144
Online API Library	144
Developer Mailing List	144
Commercial Support	144

Chapter 1. Getting Started

OTRS is a multi-platform web application framework which was originally developed for a trouble ticket system. It supports different web servers and databases.

This manual shows how to develop your own OTRS modules and applications based on the OTRS styleguides.

Development Environment

To facilitate the writing of OTRS expansion modules, the creation of a development environment is necessary.

Framework checkout (CVS)

First of all a directory must be created in which the modules can be stored. Then switch to the new directory using the command line and check them out of OTRS 2.4 or the CVS head by using the following command (loginpassword: cvs):

```
shell> cvs -d :pserver:anonymous@source.otrs.org:/home/cvs login
# for CVS Head
shell> cvs -z3 -d :pserver:anonymous@source.otrs.org:/home/cvs co otrs
# for a specific branch
shell> cvs -z3 -d :pserver:anonymous@source.otrs.org:/home/cvs co -r
rel-2_4 otrs
```

OTRS expansion modules can be checked out following the same routine. Check out the "module-tools" module, too for your development environment. It contains a number of useful tools.

To enable the CVS-OTRS it is necessary to configure it on the Apache web server and to create the Config.pm. Then the Installer.pl can be executed. The basic system is ready to run now.

Linking Expansion Modules

A clear separation between OTRS and the modules is necessary for proper developing. Particularly when using a CVS, a clear separation is crucial. In order to facilitate the OTRS access to the files, links must be created. This is done by a script in the directory module tools (to get this tools, check out the CVS module "module-tools"). Example: Linking the Calendar Module:

```
shell> ~/src/module-tools/link.pl ~/src/Calendar/ ~/src/otrs/
```

Whenever new files are added, they must be linked as described above.

To remove links from OTRS enter the following command:

```
shell> ~/src/module-tools/remove_links.pl ~/src/otrs/
```

Necessary Actions after Linking

As soon as the linking is completed, the Sysconfig must be run to register the module in OTRS. Required users, groups and roles must be created manually and access authorizations must be defined. If an additional databank table is required, this must be created manually, too. If an OPM package exists, the SQL commands can be read out to create the tables. Example:

```
shell> cat Calendar.sopm | bin/xml2sql.pl -t mysql
```

Architecture Overview

The OTRS framework is modular. The following picture shows the basic layer architecture of OTRS.

Directories

Directory	Description
bin/	commandline tools
bin/cgi-bin/	web handle
bin/cgi-bin/	fast cgi web handle
Kernel	application codebase
Kernel/Config/	configuration files
Kernel/Config/Files	configuration files
Kernel/Language	language translation files
Kernel/System/	core modules, e.g. Log, Ticket...
Kernel/Modules/	frontend modules, e.g. QueueView...
Kernel/Output/HTML/	html templates
var/	variable data
var/log	logfiles
var/cron/	cron files
var/httpd/htdocs/	htdocs directory with index.html
var/httpd/htdocs/skins/Agent/	available skins for the Agent interface
var/httpd/htdocs/skins/Customer/	available skins for the Customer interface
var/httpd/htdocs/js/	JavaScript files
scripts/	misc files
scripts/test/	unit test files
scripts/test/sample/	unit test sample data files

Files

.pl = Perl

.pm = Perl Modul

.dtl = Dynamic Template Language (html template file)

.dist = Default Templates of Files

Core Modules

Core modules are located under `$OTRS_HOME/Kernel/System/*`. This layer is for the logical work. Core modules are used to handle system routines like "lock ticket" and "create ticket". A few main core modules are:

- Kernel::System::Config (to access config options)
- Kernel::System::Log (to log into OTRS log backend)
- Kernel::System::DB (to access the database backend)
- Kernel::System::Auth (to check a user authentication)
- Kernel::System::User (to manage users)
- Kernel::System::Group (to manage groups)
- Kernel::System::Email (for sending emails)

For more information, see: <http://dev.otrs.org/>

Frontend Handle

The interface between the browser, web server and the frontend modules. A frontend module can be used via the http-link.

<http://localhost/otrs/index.pl?Action=Modul> []

Frontend Modules

Frontend modules are located under `"$OTRS_HOE/Kernel/Modules/*.pm"`. There are two public functions in there - "new()" and "run()" - which are accessed from the Frontend Handle (e.g. `index.pl`).

"new()" is used to create a frontend module object. The Frontend Handle provides the used frontend module with the basic framework objects. These are, for example: ParamObject (to get formular params), DBObject (to use existing database connects), LayoutObject (to use templates and other html layout functions), ConfigObject (to access config settings), LogObject (to use the framework log system), UserObject (to get the user functions from the current user), GroupObject (to get the group functions).

For more information on core modules see: <http://dev.otrs.org/>

CMD Frontend

The CMD (Command) Frontend is like the Web Frontend Handle and the Web Frontend Module in one (just without the LayoutObject) and uses the core modules for some actions in the system.

Database

The database interface supports different databases.

For the OTRS data model please refer to the files in your /doc directory. Alternatively you can look at the data model on our CVS server: <http://source.otrs.org/viewvc.cgi/otrs/doc/otrs-database.png?view=markup>.

Chapter 2. Hacking OTRS

In this chapter, we'll take a closer look at how OTRS works, how to extend it and how to publish your OTRS extensions.

How it Works

Config Mechanism

Default Config

There are different default config files. The main one, which comes with the framework, is:

```
Kernel/Config/Defaults.pm
```

This file should be left untouched as it is automatically updated on framework updates. There is also a sub directory where you can store the default config files for your own modules. These files are used automatically.

The directory is located under:

```
$OTRS_HOME/Kernel/Config/Files/*.pm
```

And could look as follows:

```
Kernel/config/Files/Calendar.pm
```

```
# module reg and nav bar
$self->{'Frontend::Module'}->{'AgentCalendar'} = {
    Description => 'Calendar',
    NavBarName => 'Ticket',
    NavBar => [
        {
            Description => 'Calendar',
            Name => 'Calendar',
            Image => 'calendar.png',
            Link => 'Action=AgentCalendar',
            NavBar => 'Ticket',
            Prio => 5000,
            AccessKey => 'c',
        },
    ],
};

# show online customers
$self->{'Frontend::NotifyModule'}->{'80-ShowCalendarEvents'} = {
    Module => 'Kernel::Output::HTML::NotificationCalendar',
};
```

Custom Config

If you want to change a config option, copy it to

Kernel/Config.pm

and set the new option. This file will be read out last and so all default config options are overwritten with your settings.

This way it is easy to handle updates - you just need the Kernel/Config.pm.

Accessing Config Options

You can read and write (for one request) the config options via the core module "Kernel::Config". The config object is a base object and thus available in each Frontend Module.

If you want to access a config option:

```
my $ConfigOption = $Self->{ConfigObject}->Get('Prefix::Option');
```

If you want to change a config option at runtime and just for this one request/process:

```
$Self->{ConfigObject}->Set (
    Key => 'Prefix::Option'
    Value => 'SomeNewValue',
);
```

XML Config Options

XML config files are located under:

```
$OTRS_HOME/Kernel/Config/Files/*.xml
```

Each config file has the following layout:

```
<?xml version="1.0" encoding="utf-8" ?>
<otrs_config version="1.0" init="Changes">

    <!-- config items will be here -->

</otrs_config>
```

The "init" attribute describes where the config options should be loaded. There are different levels available and will be loaded/overloaded in the following order: "Framework" (for framework settings e. g. session option), "Application" (for application settings e. g. ticket options), "Config" (for extensions to existing applications e. g. ITSM options) and "Changes" (for custom development e. g. to overwrite framework or ticket options).

If you want to add config options, here is an example:

```
<ConfigItem Name="Ticket::Hook" Required="1" Valid="1"
    ConfigLevel="300">
    <Description Lang="en">The identifier for a ticket. The default is
    Ticket#.</Description>
```

```
<Description Lang="de">Ticket-Identifikator. Als Standard wird
Ticket# verwendet.</Description>
<Group>Ticket</Group>
<SubGroup>Core::Ticket</SubGroup>
<Setting>
  <String Regex="">Ticket#</String>
</Setting>
</ConfigItem>
```

If "required" is set to "1", the config variable is included and cannot be disabled.

If "valid" is set to "1", the config variable is active. If it is set to "0", the config variable is inactive.

If the optional attribute "ConfigLevel" is set, the config variable might not be edited by the administrator, depending on his own config level. The config variable "ConfigLevel" sets the level of technical experience of the administrator. It can be 100 (Expert), 200 (Advanced) or 300 (Beginner). As a guideline which config level should be given to an option, it is recommended that all options having to do with the configuration of external interaction, like Sendmail, LDAP, SOAP, and others should get a config level of at least 200 (Advanced).

The config variable is defined in the "setting" element.

Types of XML Config Variables

The XML config settings support various types of variables.

String

A config element for numbers and single-line strings. Checking the validity with a regex is possible. The check attribute checks elements on the file system. This contains files and directories.

```
<Setting>
  <String Regex="" Check="File"></String>
</Setting>
```

Textarea

A config element for multiline text.

```
<Setting>
  <TextArea Regex=""></TextArea>
</Setting>
```

Options

This config element offers preset values as a pull-down menu.

```
<Setting>
  <Option SelectedID="Key">
```

```
        <Item Key=""></Item>
        <Item Key=""></Item>
    </Option>
</Setting>
```

Array

With this config element arrays can be displayed.

```
<Setting>
  <Array>
    <Item></Item>
    <Item></Item>
  </Array>
</Setting>
```

Hash

With this config element hashes can be displayed.

```
<Setting>
  <Hash>
    <Item Key=""></Item>
    <Item Key=""></Item>
  </Hash>
</Setting>
```

Hash with SubArray, SubHash

A hash can contain content, arrays or hashes.

```
<Setting>
  <Hash>
    <Item Key=""></Item>
    <Item Key="">
      <Hash>
        <Item Key=""></Item>
        <Item Key=""></Item>
      </Hash>
    </Item>
    <Item Key="">
      <Array>
        <Item></Item>
        <Item></Item>
      </Array>
    </Item>
    <Item Key=""></Item>
  </Hash>
</Setting>
```

FrontendModuleReg (NavBar)

Module registration for Agent Interface.

```
<Setting>
  <FrontendModuleReg>
    <Group>group1</Group>
    <Group>group2</Group>
    <Description>Logout</Description>
    <Title></Title>
    <NavBarName></NavBarName>
    <NavBar>
      <Description>Logout</Description>
      <Name>Logout</Name>
      <Image>exit.png</Image>
      <Link>Action=Logout</Link>
      <NavBar></NavBar>
      <Type></Type>
      <Block>ItemPre</Block>
      <AccessKey>l</AccessKey>
      <Prio>100</Prio>
    </NavBar>
  </FrontendModuleReg>
</Setting>
```

FrontendModuleReg (NavBarModule)

Module registration for Admin Interface

```
<Setting>
  <FrontendModuleReg>
    <Group>admin</Group>
    <Group>admin2</Group>
    <Description>Admin</Description>
    <Title>User</Title>
    <NavBarName>Admin</NavBarName>
    <NavBarModule>
      <Module>Kernel::Output::HTML::NavBarModuleAdmin</Module>
      <Name>Users</Name>
      <Block>Block1</Block>
      <Prio>100</Prio>
    </NavBarModule>
  </FrontendModuleReg>
</Setting>
```

Database Mechanism

OTRS comes with a database layer that supports different databases.

How it works

The database layer (Kernel::System::DB) has two input options: SQL and XML.

SQL

The SQL interface should be used for normal database actions (SELECT, INSERT, UPDATE, ...). It can be used like a normal Perl DBI interface.

INSERT/UPDATE/DELETE

```
$Self->{DBObject}->Do(  
    SQL=> "INSERT INTO table (name, id) VALUES ('SomeName', 123)",  
);  
  
$Self->{DBObject}->Do(  
    SQL=> "UPDATE table SET name = 'SomeName', id = 123",  
);  
  
$Self->{DBObject}->Do(  
    SQL=> "DELETE FROM table WHERE id = 123",  
);
```

SELECT

```
my $SQL = "SELECT id FROM table WHERE tn = '123'";  
  
$Self->{DBObject}->Prepare(SQL => $SQL, Limit => 15);  
  
while (my @Row = $Self->{DBObject}->FetchrowArray()) {  
    $Id = $Row[0];  
}  
return $Id;
```

Note

Take care to use Limit as param and not in the SQL string because not all databases support LIMIT in SQL strings.

```
my $SQL = "SELECT id FROM table WHERE tn = ? AND group = ?";  
  
$Self->{DBObject}->Prepare(  
    SQL    => $SQL,  
    Limit => 15,  
    Bind   => [ $Tn, $Group ],  
);  
  
while (my @Row = $Self->{DBObject}->FetchrowArray()) {  
    $Id = $Row[0];  
}
```

```
return $Id;
```

Note

Use the Bind attribute where ever you can, especially for long statements. If you use Bind you do not need the function Quote().

QUOTE

String:

```
my $QuotedString = $Self->{DBObject}->Quote("It's a problem!");
```

Integer:

```
my $QuotedInteger = $Self->{DBObject}->Quote('123', 'Integer');
```

Number:

```
my $QuotedNumber = $Self->{DBObject}->Quote('21.35', 'Number');
```

Note

Please use the Bind attribute instead of Quote() where ever you can.

XML

The XML interface should be used for INSERT, CREATE TABLE, DROP TABLE and ALTER TABLE. As this syntax is different from database to database, using it makes sure that you write applications that can be used in all of them.

Note

The <Insert> has changed in >=2.2. Values are now used in content area (not longer in attribut Value).

INSERT

```
<Insert Table="some_table">
  <Data Key="id">1</Data>
  <Data Key="description" Type="Quote">exploit</Data>
</Insert>
```

CREATE TABLE

Possible data types are: BIGINT, SMALLINT, INTEGER, VARCHAR (Size=1-1000000), DATE (Format: yyyy-mm-dd hh:mm:ss) and LONGBLOB.

```

<TableCreate Name="calendar_event">
  <Column Name="id" Required="true" PrimaryKey="true"
  AutoIncrement="true" Type="BIGINT"/>
  <Column Name="title" Required="true" Size="250" Type="VARCHAR"/>
  <Column Name="content" Required="false" Size="250" Type="VARCHAR"/
>
  <Column Name="start_time" Required="true" Type="DATE"/>
  <Column Name="end_time" Required="true" Type="DATE"/>
  <Column Name="owner_id" Required="true" Type="INTEGER"/>
  <Column Name="event_status" Required="true" Size="50"
Type="VARCHAR"/>
  <Index Name="calendar_event_title">
    <IndexColumn Name="title"/>
  </Index>
  <Unique Name="calendar_event_title">
    <UniqueColumn Name="title"/>
  </Unique>
  <ForeignKey ForeignTable="users">
    <Reference Local="owner_id" Foreign="id"/>
  </ForeignKey>
</TableCreate>

```

DROP TABLE

```

<TableDrop Name="calendar_event"/>

```

ALTER TABLE

The following shows an example of add, change and drop columns.

```

<TableAlter Name="calendar_event">
  <ColumnAdd Name="test_name" Type="varchar" Size="20" Required="1"/
>
  <ColumnChange NameOld="test_name" NameNew="test_title"
Type="varchar" Size="30" Required="1"/>
  <ColumnChange NameOld="test_title" NameNew="test_title"
Type="varchar" Size="100" Required="0"/>
  <ColumnDrop Name="test_title"/>
  <IndexCreate Name="index_test3">
    <IndexColumn Name="test3"/>
  </IndexCreate>
  <IndexDrop Name="index_test3"/>
  <UniqueCreate Name="uniq_test3">
    <UniqueColumn Name="test3"/>

```

```
</UniqueCreate>  
  
<UniqueDrop Name="uniq_test3"/>  
</TableAlter>
```

The next shows an example how to rename a table.

```
<TableAlter NameOld="calendar_event" NameNew="calendar_event_new"/>
```

Code to process XML

```
my @XMLARRAY = @{$Self->ParseXML(String => $XML)};  
  
my @SQL = $Self->{DBObject}->SQLProcessor(  
    Database => \@XMLARRAY,  
);  
push(@SQL, $Self->{DBObject}->SQLProcessorPost());  
  
for (@SQL) {  
    $Self->{DBObject}->Do(SQL => $_);  
}
```

Database Drivers

The database drivers are located under `$OTRS_HOME/Kernel/System/DB/*.pm`.

Supported Databases

- MySQL
- PostgreSQL
- Oracle
- MSSQL
- DB2

Log Mechanism

OTRS comes with a log backend that can be used for application logging and debugging.

Use and Syntax

All module layers have ready-made Log Objects which can be used by

```
$Self->{LogObject}->Log (
```

```
    Priority => 'error',  
    Message => 'Need something!',  
);
```

Example

The following example shows how to use the log mechanism without a module layer.

```
use Kernel::Config;  
use Kernel::System::Encode;  
use Kernel::System::Log;  
  
my $ConfigObject = Kernel::Config->new();  
my $EncodeObject = Kernel::System::Encode->new(  
    ConfigObject => $ConfigObject,  
);  
my $LogObject    = Kernel::System::Log->new(  
    ConfigObject => $ConfigObject,  
);  
  
$Self->{LogObject}->Log(  
    Priority => 'error',  
    Message => 'Need something!',  
);
```

Skins

Since OTRS 3.0, the visual appearance of OTRS is controlled by "skins".

A skin is a set of CSS and image files, which together control how the GUI is presented to the user. Skins do not change the HTML content that is generated by OTRS (this is what "Themes" do), but they control how it is displayed. With the help of modern CSS standards it is possible to change the display thoroughly (e.g. repositioning parts of dialogs, hiding elements, ...).

Skin Basics

All skins are in `$OTRS_HOME/var/httpd/htdocs/skins/$SKIN_TYPE/$SKIN_NAME`. There are two types of skins: agent and customer skins.

Each of the agents can select individually, which of the installed agent skins they want to "wear".

For the customer interface, a skin has to be selected globally with the config setting `Loader::Customer::SelectedSkin`. All customers will see this skin.

How skins are loaded

It is important to note that the "default" skin will *always* be loaded *first*. If the agent selected another skin than the "default" one, then the other one will be loaded only *after* the default skin. By "loading" here we mean that OTRS will put tags into the HTML content which cause the CSS files to be loaded by the browser. Let's see an example of this:

```
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css-
cache/CommonCSS_179376764084443c181048401ae0e2ad.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/ivory/css-
cache/CommonCSS_e0783e0c2445ad9cc59c35d6e4629684.css" />
```

Here it can clearly be seen that the default skin is loaded first, and then the custom skin specified by the agent. In this example, we see the result of the activated loader (`Loader::Enabled` set to 1), which gathers all CSS files, concatenates and minifies them and serves them as one chunk to the browser. This saves bandwidth and also reduces the number of HTTP requests. Let's see the same example with the Loader turned off:

```
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.Reset.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.Default.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.Header.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.OverviewControl.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.OverviewSmall.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.OverviewMedium.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.OverviewLarge.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.Footer.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.Grid.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.Form.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.Table.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.Widget.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.WidgetMenu.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.TicketDetail.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.Tooltip.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.Dialog.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.Print.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.Agent.CustomerUser.GoogleMaps.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/default/css/
Core.Agent.CustomerUser.OpenTicket.css" />
<link rel="stylesheet" href="/otrs-web/skins/Agent/ivory/css/
Core.Dialog.css" />
```

Here we can better see the individual files that come from the skins.

There are different types of CSS files: common files which must always be loaded, and "module-specific" files which are only loaded for special modules within the OTRS framework.

In addition, it is possible to specify CSS files which only must be loaded on IE7 or IE8 (in the case of the customer interface, also IE6). This is unfortunate, but it was not possible to develop a modern GUI on these browsers without having special CSS for them.

For details regarding the CSS file types, please see the section on the Loader.

For each HTML page generation, the loader will first take all configured CSS files from the default skin, and then for each file look if it is also available in a custom skin (if a custom skin is selected) and load them after the default files.

That means a) that CSS files in custom skins need to have the same names as in the default skins, and b) that a custom skin does not need to have all files of the default skin. That is the big advantage of loading the default skin first: a custom skin has all default CSS rules present and only needs to change those which should result in a different display. That can often be done in a single file, like in the example above.

Another effect of this is that you need to be careful to overwrite all default CSS rules in your custom skins that you want to change. Let's see an example:

```
.Header h1 {
    font-weight: bold;
    color: #000;
}
```

This defines special headings inside of the `.Header` element as bold, black text. Now if you want to change that in your skin to another color and normal text, it is not enough to write

```
.Header h1 {
    color: #F00;
}
```

Because the original rule for `font-weight` still applies. You need to override it explicitly:

```
.Header h1 {
    font-weight: normal;
    color: #F00;
}
```

Creating a New Skin

In this section, we will be creating a new agent skin which replaces the default OTRS background color (white) with a custom company color (light grey) and the default logo by a custome one. Also we will configure that skin to be the one which all agents will see by default.

There are only two simple steps we need to take to achieve this goal:

- create the skin files
- configure the new logo and
- make the skin known to the OTRS system.

Let's start by creating the files needed for our new skin. First of all, we need to create a new folder for this skin (we'll call it "custom"). This folder will be `$OTRS_HOME/var/httpd/htdocs/skins/Agent/custom`.

In there, we need to place the new CSS file in a new directory `css` which defines the new skin's appearance. We'll call it `Core.Default.css` (remember that it must have the same name as one of the files in the "default" skin). This is the code needed for the CSS file:

```
body {
    background-color: #c0c0c0; /* not very beautiful but it meets our
    purpose */
}
```

Now follows the second step, adding a new logo and making the new skin known to the OTRS system. For this, we first need to place our custom logo (e.g. `logo.png`) in a new directory (e.g. `img`) in our skin directory. Then we need to create a new config file `$OTRS_HOME/Kernel/Config/Files/CustomSkin.xml`, which will contain the needed settings as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<otrs_config version="1.0" init="Framework">
  <ConfigItem Name="AgentLogo" Required="0" Valid="1">
    <Description Translatable="1">The logo shown in the header of
    the agent interface. The URL to the image must be a relative URL to
    the skin image directory.</Description>
    <Group>Framework</Group>
    <SubGroup>Frontend::Agent</SubGroup>
    <Setting>
      <Hash>
        <Item Key="URL">skins/Agent/custom/img/logo.png</Item>
        <Item Key="StyleTop">13px</Item>
        <Item Key="StyleRight">75px</Item>
        <Item Key="StyleHeight">67px</Item>
        <Item Key="StyleWidth">244px</Item>
      </Hash>
    </Setting>
  </ConfigItem>
  <ConfigItem Name="Loader::Agent::Skin###100-custom" Required="0"
  Valid="1">
    <Description Translatable="1">Custom skin for the development
    manual.</Description>
    <Group>Framework</Group>
    <SubGroup>Frontend::Agent</SubGroup>
    <Setting>
```

```
<Hash>
  <Item Key="InternalName">custom</Item>
  <Item Key="VisibleName">Custom</Item>
  <Item Key="Description">Custom skin for the
development manual.</Item>
  <Item Key="HomePage">www.yourcompany.com</Item>
</Hash>
</Setting>
</ConfigItem>
</otrs_config>
```

To make this configuration active, we need to navigate to the SysConfig module in the admin area of OTRS (alternatively, you can run the script `$OTRS_HOME/bin/otrs.RebuildConfig.pl`). This will regenerate the Perl cache of the XML configuration files, so that our new skin is now known and can be selected in the system. To make it the default skin that new agents see before they made their own skin selection, edit the SysConfig setting "Loader::Agent::DefaultSelectedSkin" and set it to "custom".

In conclusion: to create a new skin in OTRS, we had to place the new logo file, and create one CSS and one XML file, resulting in three new files:

```
$OTRS_HOME/Kernel/Config/Files/CustomSkin.xml
$OTRS_HOME/var/httpd/htdocs/skins/Agent/custom/img/custom-logo.png
$OTRS_HOME/var/httpd/htdocs/skins/Agent/custom/css/Core.Header.css
```

The CSS and JavaScript "Loader"

Starting with OTRS 3.0, the CSS and JavaScript code in OTRS grew to a large amount. To be able to satisfy both development concerns (good maintainability by a large number of separate files) and performance issues (making few HTTP requests and serving minified content without unnecessary whitespace and documentation) had to be addressed. To achieve these goals, the Loader was invented.

How it works

To put it simple, the Loader

- determines for each request precisely which CSS and JavaScript files are needed at the client side by the current application module
- collects all the relevant data
- minifies the data, removing unnecessary whitespace and documentation
- serves it to the client in only a few HTTP requests instead of many individual ones, allowing the client to cache these snippets in the browser cache
- perform these tasks in a highly performing way, utilizing the caching mechanisms of OTRS.

Of course, there is a little bit more detailed involved, but this should suffice as a first overview.

Basic Operation

With the configuration settings `Loader::Enabled::CSS` and `Loader::Enabled::JavaScript`, the loader can be turned on and off for JavaScript and CSS, respectively (it is on by default).

Warning

Because of rendering problems in Internet Explorer, the Loader cannot be turned off for CSS files for this client browser (config setting will be overridden). Up to version 8, Internet Explorer cannot handle more than 32 CSS files on a page.

To learn about how the Loader works, please turn it off in your OTRS installation with the aforementioned configuration settings. Now look at the source code of the application module that you are currently using in this OTRS system (after a reload, of course). You will see that there are many CSS files loaded in the `<head>` section of the page, and many JavaScript files at the bottom of the page, just before the closing `</body>` element.

Having the content like this in many individual files with a readable formatting makes the development much easier, and even possible at all. However, this has the disadvantage of a large number of HTTP requests (network latency has a big effect) and unnecessary content (whitespace and documentation) which needs to be transferred to the client.

The Loader solves this problem by performing the steps outlined in the short description above. Please turn on the Loader again and reload your page now. Now you can see that there are only very few CSS and JavaScript tags in the HTML code, like this:

```
<script type="text/javascript" src="/otrs30-dev-web/js/js-cache/
CommonJS_d16010491cbd4faaaeb740136a8ecbfd.js"></script>
```

```
<script type="text/javascript" src="/otrs30-dev-web/js/js-cache/
ModuleJS_b54ba9c085577ac48745f6849978907c.js"></script>
```

What just happened? During the original request generating the HTML code for this page, the Loader generated these two files (or took them from the cache) and put the shown `<script>` tags on the page which link to these files, instead of linking to all relevant JavaScript files separately (as you saw it without the loader being active).

The CSS section looks a little more complicated:

```
<link rel="stylesheet" type="text/css" href="/
otrs30-dev-web/skins/Agent/default/css-cache/
CommonCSS_00753c78c9be7a634c70e914486bfbad.css" />
```

```
<!--[if IE 7]>
```

```
<link rel="stylesheet" type="text/css" href="/
otrs30-dev-web/skins/Agent/default/css-cache/
CommonCSS_IE7_59394a0516ce2e7359c255a06835d31f.css" />
<![endif]-->
```

```
<!--[if IE 8]>
```

```
<link rel="stylesheet" type="text/css" href="/
otrs30-dev-web/skins/Agent/default/css-cache/
CommonCSS_IE8_ff58bd010ef0169703062b6001b13ca9.css" />
<![endif]-->
```

The reason is that Internet Explorer 7 and 8 need special treatment in addition to the default CSS because of their lacking support of web standard technologies. So we have some normal CSS that is loaded in all browsers, and some special CSS that is inside of so-called "conditional comments" which cause it to be loaded *only* by Internet Explorer 7/8. All other browsers will ignore it.

Now we have outlined how the loader works. Let's look at how you can utilize that in your own OTRS extensions by adding configuration data to the loader, telling it to load additional or alternative CSS or JavaScript content.

Configuring the Loader: JavaScript

To be able to operate correctly, the Loader needs to know which content it has to load for a particular OTRS application module. First, it will look for JavaScript files which *always* have to be loaded, and then it looks for special files which are only relevant for the current application module.

Common JavaScript

The list of JavaScript files to be loaded is configured in the configuration settings `Loader::Agent::CommonJS` (for the agent interface) and `Loader::Customer::CommonJS` (for the customer interface).

These settings are designed as hashes, so that OTRS extensions can add their own hash keys for additional content to be loaded. Let's look at an example:

```
<ConfigItem Name="Loader::Agent::CommonJS###000-Framework"
Required="1" Valid="1">
  <Description Translatable="1">List of JS files to always be loaded
  for the agent interface.</Description>
  <Group>Framework</Group>
  <SubGroup>Core::Web</SubGroup>
  <Setting>
    <Array>
      <Item>thirdparty/json/json2.js</Item>
      <Item>thirdparty/jquery-1.4.2/jquery.js</Item>
      ...
      <Item>Core.App.js</Item>
      <Item>Core.Agent.js</Item>
      <Item>Core.Agent.Search.js</Item>
    </Array>
  </Setting>
</ConfigItem>
```

This is the list of JavaScript files which always need to be loaded for the agent interface of OTRS.

To add new content which is supposed to be loaded always in the agent interface, just add an XML configuration file with another hash entry:

```
<ConfigItem Name="Loader::Agent::CommonJS###100-CustomPackage"
  Required="0" Valid="1">
  <Description Translatable="1">List of JS files to always be loaded
  for the agent interface for package "CustomPackage".</Description>
  <Group>Framework</Group>
  <SubGroup>Core::Web</SubGroup>
  <Setting>
    <Array>
      <Item>CustomPackage.App.js</Item>
    </Array>
  </Setting>
</ConfigItem>
```

Simple, isn't it?

Module-Specific JavaScript

Not all JavaScript is usable for all application modules of OTRS. Therefore it is possible to specify module-specific JavaScript files. Whenever a certain module is used (such as AgentDashboard), the module-specific JavaScript for this module will also be loaded. The configuration is done in the frontend module registration in the XML configurations. Again, an example:

```
<ConfigItem Name="Frontend::Module###AgentDashboard" Required="0"
  Valid="1">
  <Description Translatable="1">Frontend module registration for the
  agent interface.</Description>
  <Group>Framework</Group>
  <SubGroup>Frontend::Agent::ModuleRegistration</SubGroup>
  <Setting>
    <FrontendModuleReg>
      <Description>Agent Dashboard</Description>
      <Title></Title>
      <NavBarName>Dashboard</NavBarName>
      <NavBar>
        <Description Translatable="1"></Description>
        <Name Translatable="1">Dashboard</Name>
        <Link>Action=AgentDashboard</Link>
        <NavBar>Dashboard</NavBar>
        <Type>Menu</Type>
        <Description Translatable="1"></Description>
        <Block>ItemArea</Block>
        <AccessKey>d</AccessKey>
        <Prio>50</Prio>
      </NavBar>
      <Loader>
        <JavaScript>thirdparty/flot/excanvas.js</JavaScript>
        <JavaScript>thirdparty/flot/jquery.flot.js</
JavaScript>
```

```

        <JavaScript>Core.UI.Chart.js</JavaScript>
        <JavaScript>Core.UI.DnD.js</JavaScript>
        <JavaScript>Core.Agent.Dashboard.js</JavaScript>
    </Loader>
</FrontendModuleReg>
</Setting>
</ConfigItem>

```

It is possible to put a `<Loader>` tag in the frontend module registrations which may contain `<JavaScript>` tags, one for each file that is supposed to be loaded for this application module.

Now you have all information you need to configure the way the Loader handles JavaScript code.

There is one special case: for `ToolBarModules`, you can also add custom JavaScript files. Just add a `JavaScript` attribute to the configuration like this:

```

<ConfigItem Name="Frontend::ToolBarModule###410-
Ticket::AgentTicketEmail" Required="0" Valid="1">
    <Description Translatable="1">Toolbar Item for a shortcut.</
Description>
    <Group>Ticket</Group>
    <SubGroup>Frontend::Agent::ToolBarModule</SubGroup>
    <Setting>
        <Hash>
            <Item Key="Module">Kernel::Output::HTML::ToolBarLink</
Item>
            <Item Key="Name">New email ticket</Item>
            <Item Key="Priority">1009999</Item>
            <Item Key="Link">Action=AgentTicketEmail</Item>
            <Item Key="Action">AgentTicketEmail</Item>
            <Item Key="AccessKey">1</Item>
            <Item Key="CssClass">EmailTicket</Item>
            <Item Key="JavaScript">OTRS.Agent.CustomToolBarModule.js</
Item>
        </Hash>
    </Setting>
</ConfigItem>

```

Configuring the Loader: CSS

The loader handles CSS files very similar to JavaScript files, as described in the previous section, and extending the settings works in the same way too.

Common CSS

The way common CSS is handled is very similar to the way common JavaScript is loaded. Here, the configuration settings are called `Loader::Agent::CommonCSS` and `Loader::Customer::CommonCSS`, respectively.

However, as we already noted above, Internet Explorer 7 and 8 (and for the customer interface also 6) need special treatment. That's why there are special configuration settings for them, to specify common CSS which should only be loaded in these browsers. The respective

settings are Loader::Agent::CommonCSS::IE7, Loader::Agent::CommonCSS::IE8, Loader::Customer::CommonCSS::IE6, Loader::Customer::CommonCSS::IE7 and Loader::Customer::CommonCSS::IE8.

An example:

```
<ConfigItem Name="Loader::Agent::CommonCSS::IE8###000-Framework"
Required="1" Valid="1">
  <Description Translatable="1">List of IE8-specific CSS files to
always be loaded for the agent interface.</Description>
  <Group>Framework</Group>
  <SubGroup>Core::Web</SubGroup>
  <Setting>
    <Array>
      <Item>Core.OverviewSmall.IE8.css</Item>
    </Array>
  </Setting>
</ConfigItem>
```

This is the list of common CSS files for the agent interface which should only be loaded in Internet Explorer 8.

Module-Specific CSS

Module-specific CSS is handled very similar to the way module-specific JavaScript is handled. It is also configured in the frontend module registrations. Example:

```
<ConfigItem Name="Frontend::Module###Admin" Required="0" Valid="1">
  <Description Translatable="1">Frontend module registration for the
agent interface.</Description>
  <Group>Framework</Group>
  <SubGroup>Frontend::Admin::ModuleRegistration</SubGroup>
  <Setting>
    <FrontendModuleReg>
      <Group>admin</Group>
      <Description>Admin-Area</Description>
      <Title></Title>
      <NavBarName>Admin</NavBarName>
      <NavBar>
        <Type>Menu</Type>
        <Description Translatable="1"></Description>
        <Block>ItemArea</Block>
        <Name Translatable="1">Admin</Name>
        <Link>Action=Admin</Link>
        <NavBar>Admin</NavBar>
        <AccessKey>a</AccessKey>
        <Prio>10000</Prio>
      </NavBar>
      <NavBarModule>
        <Module>Kernel::Output::HTML::NavBarModuleAdmin</
Module>
      </NavBarModule>
```

```

        <Loader>
            <CSS>Core.Agent.Admin.css</CSS>
            <CSS_IE7>Core.Agent.AdminIE7.css</CSS_IE7>
            <JavaScript>Core.Agent.Admin.SysConfig.js</JavaScript>
        </Loader>
    </FrontendModuleReg>
</Setting>
</ConfigItem>

```

Here we have a module (the admin overview page of the agent interface) which has special JavaScript, normal CSS (tagname `<CSS>`) and special CSS for Internet Explorer 7 (tagname `<CSS_IE7>`). All of these need to be loaded in addition to the common JavaScript and CSS defined for the agent interface.

It is also possible to specify module-specific CSS for Internet Explorer 8 (tagname `<CSS_IE8>`) and, in the case of the customer interface, for Internet Explorer 6 (tagname `<CSS_IE6>`).

There is one special case: for `ToolBarModules`, you can also add custom CSS files. Just add a `CSS`, `CSS_IE7` or `CSS_IE8` attribute to the configuration like this:

```

<ConfigItem Name="Frontend::ToolBarModule###410-
Ticket::AgentTicketEmail" Required="0" Valid="1">
    <Description Translatable="1">Toolbar Item for a shortcut.</
Description>
    <Group>Ticket</Group>
    <SubGroup>Frontend::Agent::ToolBarModule</SubGroup>
    <Setting>
        <Hash>
            <Item Key="Module">Kernel::Output::HTML::ToolBarLink</
Item>
            <Item Key="Name">New email ticket</Item>
            <Item Key="Priority">1009999</Item>
            <Item Key="Link">Action=AgentTicketEmail</Item>
            <Item Key="Action">AgentTicketEmail</Item>
            <Item Key="AccessKey">l</Item>
            <Item Key="CssClass">EmailTicket</Item>
            <Item Key="CSS">OTRS.Agent.CustomToolBarModule.css</Item>
            <Item
Key="CSS_IE7">OTRS.Agent.CustomToolBarModule.IE7.css</Item>
        </Hash>
    </Setting>
</ConfigItem>

```

Templating Mechanism

Internally, OTRS uses a templating mechanism to dynamically generate its HTML pages (and other content), while keeping the program logic (Perl) and the presentation (HTML) separate. Typically, a frontend module will use an own template file, pass some data to it and return the rendered result to the user.

The template files are located at: `$OTRS_HOME/Kernel/Output/HTML/Standard/*.dtl`

Inside of these templates, a set of commands for data manipulation, localization and simple logical structures can be used. This section describes these commands and shows how to use them in templates.

Template Commands

Data Manipulation Commands

In templates, dynamic data must be inserted, quoted etc. This section lists the relevant commands to do that.

`$Data{""}`

If data parameters are given to the templates by the application module, these data can be output to the template. `$Data` is the most simple, but also most dangerous one. It will insert the data parameter whose name is specified inside of the `{""}` into the template as it is, without any HTML quoting.

Warning

Because of the missing HTML quoting, this can result in security problems. Never output data that was input by a user without quoting in HTML context. The user could - for example - just insert a `<script>` tag, and it would be output on the HTML page generated by OTRS.

Whenever possible, use `$QData{""}` (in HTML) or `$LQData{""}` (in Links) instead.

Example: Whenever we generate HTML in the application, we need to output it to the template without HTML quoting, like `<select>` elements, which are generated by the function `Layout::BuildSelection` in OTRS.

```
<label for="Dropdown">Example Dropdown</label>
$Data{"DropdownString"}
```

`$QData{""}`

This command has the same function as `$Data{""}`, but it performs HTML quoting on the data as it is inserted to the template.

```
The name of the author is $QData{"Name"}.
```

It's also possible specify a maximum length for the value. If, for example, you just want to show 8 characters of a variable (result will be "SomeName[...]"). use the following:

```
The first 20 characters of the author's name: $QData{"Name","20"}.
```

`$LQData{""}`

This command has the same function as `$Data{""}`, but it performs URL encoding [<http://en.wikipedia.org/wiki/Percent-encoding>] on the data as it is inserted to the template. This should

be used to output single parameter names or values of URLs, to prevent security problems. It cannot be used for complete URLs because it will also mask =, for example.

```
<a href="$Env{"Baselink"};Location=$LQData{"File"}">
$QData{"File","110"}</a>
```

\$Env{""}

Inserts the environment variable with the name specified in {}. Some examples:

```
The current user name is: $Env{"UserFirstname"}
```

Some other common predefined variables are:

```
$Env{"SessionID"} --> the current session id
$Env{"Time"} --> the current time e. g. Thu Dec 27 16:00:55 2001
$Env{"CGIHandle"} --> the current CGI handle e. g. index.pl
$Env{"UserCharset"} --> the current site charset e. g. iso-8859-1
$Env{"Baselink"} --> the baselink --> index.pl?SessionID=...
$Env{"UserFirstname"} --> e. g. Dirk $Env{"UserFirstname"}
$Env{"UserLogin"} --> e. g. mgg@x11.org
$Env{"UserIsGroup[users]"} = Yes --> user groups (useful for own
links)
$Env{"UserIsGroup[admin]"} = Yes $Env{"Action"} --> the current action
```

Warning

Because of the missing HTML quoting, this can result in security problems. Never output data that was input by a user without quoting in HTML context. The user could - for example - just insert a `<script>` tag, and it would be output on the HTML page generated by OTRS.

Whenever possible, use `$QEnv{ "" }` instead.

\$QEnv{""}

Works like `$Env{ "" }`, but performs HTML encoding when the data is inserted to the template.

```
The current user name is: $QEnv{"UserFirstname"}
```

\$Config{""}

With this tag you can insert config variables into the template. Let's see an example Kernel/Config.pm:

```
[Kernel/Config.pm]
# FQDN
# (Full qualified domain name of your system.)
```

```
$Self->{FQDN} = 'otrs.example.com';
# AdminEmail
# (Email of the system admin.)
$Self->{AdminEmail} = 'admin@example.com';
[...]
```

To output values from it in the template, use:

```
The hostname is '$Config{"FQDN"}'
The admin email address is '$Config{"AdminEmail"}'
```

Warning

Because of the missing HTML quoting, this can result in security problems.

Whenever possible, use `$Quote{"$Config{""}"}`.

\$Quote{""}

This tag can be used to perform quoting on HTML strings, when no other quoting is possible.

```
$Quote{"$Config{"ProductName"}"} ($Quote{"$Config{"Ticket::Hook"}"})
```

It's also possible specify a maximum length for the value. If, for example, you just want to show 8 characters of a variable (result will be "Some lon[...]"). use the following:

```
$Quote{"Some long long string", "8"})
```

Localization Commands

\$Text{""}

Translates the enclosed string into the current user's selected language and performs HTML quoting on the resulting string. If no translation is found, the original string will be used.

```
Translate this text: $Text{"Help"}
```

When translating data coming from the application, use `$Data` inside of `$Text`, not `$QData`, to prevent double quoting:

```
Translate data from the application: $Text{"$Data{"Type"}"}
```

You can also specify parameters (%s) inside of the string which should be replaced with other data:

Translate this text and insert the given data: `$Text{"Change %s settings", "$Data{"Type"}"}`

`$JSText{""}`

Works in the same way as `$Text{""}`, but does not perform HTML encoding but JavaScript string escaping instead (all `'` characters will be encoded as `\'`. So with the help of this tag you can make sure that even dynamic strings will not break your JavaScript code.

```
window.alert('$JSText{"Some message's content"}');

// after the command was replaced in the template, this will
// result in (for an English speaking agent):

window.alert('Some message\'s content');
```

Make sure to use `'` as string delimiter for strings where you want to use `$JSText` inside.

`$TimeLong{""}`

Inserts a localized date/time stamp (including a possible time zone difference of the current agent).

In different cultural areas, different convention for date and time formatting are used. For example, what is the 01.02.2010 in Germany, would be 02/01/2010 in the USA. `$Time{""}` abstracts this away from the templates. Let's see an example:

```
# from AgentTicketHistory.dtl
$TimeLong{"$Data{"CreateTime"}"}

# Result for US English locale:
06/09/2010 15:45:41
```

First, the data is inserted from the application module with `$Data`. Here always an ISO UTC timestamp (2010-06-09 15:45:41) must be passed as data to `$TimeLong{""}`. Then `$TimeLong{""}` will take that data and output it according to the date/time definition of the current locale.

The data passed to `$TimeLong{""}` must be UTC. If a time zone offset is specified for the current agent, it will be applied to the UTC timestamp before the output is generated.

`$TimeShort{""}`

Works like `$TimeLong{""}`, but does not output the seconds.

```
$TimeShort{"$Data{"CreateTime"}"}

# Result for US English locale:
```

06/09/2010 15:45

\$Date{""}

Works like `$TimeLong{""}`, but outputs only the date, not the time.

```
$Date{"$Data{"CreateTime"}}}
```

```
# Result for US English locale:  
06/09/2010
```

Template Processing Commands

Comment

The `dtl` comment starts with a `#` at the beginning of a line and will not be shown in the html output. This can be used both for commenting the DTL (=Template) code or for disabling parts of it.

```
# this section is temporarily disabled  
# <div class="AsBlock">  
#   <a href="...">link</a>  
# </div>
```

\$Include{""}

Includes another template file into the current one. The included file may also contain template commands.

```
# include Copyright.dtl  
$Include{"Copyright"}
```

dtl:block

With this command, one can specify parts of a template file as a block. This block needs to be explicitly filled with a function call from the application, to be present in the generated output. The application can call the block 0 (it will not be present in the output), 1 or more times (each with possibly a different set of data parameters passed to the template).

One common use case is the filling of a table with dynamic data:

```
<table class="DataTable">  
  <thead>  
    <tr>  
      <th>$Text{"Name"}</th>  
      <th>$Text{"Type"}</th>  
      <th>$Text{"Comment"}</th>  
      <th>$Text{"Valid"}</th>
```

```

        <th>$Text{"Changed"}</th>
        <th>$Text{"Created"}</th>
    </tr>
</thead>
<tbody>
<!-- dtl:block:NoDataFoundMsg -->
    <tr>
        <td colspan="6">
            $Text{"No data found."}
        </td>
    </tr>
<!-- dtl:block:NoDataFoundMsg -->
<!-- dtl:block:OverviewResultRow -->
    <tr>
        <td><a class="AsBlock" href="$Env{"Baselink"}Action=
$Env{"Action"};Subaction=Change;ID=$LQData{"ID"}">$QData{"Name"}</a></
td>
        <td>$Text{"$Data{"TypeName"}"}</td>
        <td title="$QData{"Comment"}">$QData{"Comment", "20"}</td>
        <td>$Text{"$Data{"Valid"}"}</td>
        <td>$TimeShort{"$QData{"ChangeTime"}"}</td>
        <td>$TimeShort{"$QData{"CreateTime"}"}</td>
    </tr>
<!-- dtl:block:OverviewResultRow -->
</tbody>
</table>

```

The surrounding table with the header is always generated. If no data was found, the block `NoDataFoundMsg` is called once, resulting in a table with one data row with the message "No data found."

If data was found, for each row there is one function call made for the block `OverviewResultRow` (each time passing in the data for this particular row), resulting in a table with as many data rows as results were found.

Let's look at how the blocks are called from the application module:

```

my %List = $Self->{StateObject}->StateList(
    UserID => 1,
    Valid => 0,
);

# if there are any states, they are shown
if (%List) {

    # get valid list
    my %ValidList = $Self->{ValidObject}->ValidList();
    for ( sort { $List{$a} cmp $List{$b} } keys %List ) {

        my %Data = $Self->{StateObject}->StateGet( ID => $_, );
        $Self->{LayoutObject}->Block(
            Name => 'OverviewResultRow',
            Data => {

```

```

        Valid => $ValidList{ $Data{ValidID} },
        %Data,
    },
    );
}
}

# otherwise a no data found msg is displayed
else {
    $Self->{LayoutObject}->Block(
        Name => 'NoDataFoundMsg',
        Data => {},
    );
}
}

```

Note how the blocks have both their name and an optional set of data passed in as separate parameters to the block function call. Data inserting commands inside a block always need the data provided to the block function call of this block, not the general template rendering call.

For details, please refer to the documentation of `Kernel::Output::HTML::Layout` on dev.otrs.org [<http://dev.otrs.org>].

dtl:js_on_document_complete

Marks JavaScript code which should be executed after all CSS, JavaScript and other external content has been loaded and the basic JavaScript initialization was finished. Again, let's look at an example:

```

<form title="$Text{"Move ticket to a different queue"}"
  action="$Env{"CGIHandle"}" method="get">
  <input type="hidden" name="Action" value="AgentTicketMove"/>
  <input type="hidden" name="QueueID" value="$QData{"QueueID"}"/>
  <input type="hidden" name="TicketID" value="$QData{"TicketID"}"/>
  <label for="DestQueueID" class="InvisibleText">$Text{"Change
  queue"}:</label>
  $Data{"MoveQueuesStrg"}
</form>
<!-- dtl:js_on_document_complete -->
<script type="text/javascript">
  $('#DestQueueID').bind('change', function (Event) {
    $(this).closest('form').submit();
  });
</script>
<!-- dtl:js_on_document_complete -->

```

This snippet creates a small form and puts an onchange-Handler on the `<select>` element which causes and automatical form submit.

Why is it necessary to enclose the JavaScript code in `dtl:js_on_document_complete`? Starting with OTRS 3.0, JavaScript loading was moved to the footer part of the page for performance reasons. This means that within the `<body>` of the page, no JavaScript libraries are loaded yet. With `dtl:js_on_document_complete` you can make sure that this JavaScript

is moved to a part of the final HTML document, where it will be executed only after the entire external JavaScript and CSS content has been successfully loaded and initialized.

Inside the `dtl:js_on_document_complete` block, you can use `<script>` tags to enclose your JavaScript code, but you do not have to do so. It may be beneficial because it will enable correct syntax highlighting in IDEs which support it.

Using a template file

Ok, but how to actually process a template file and generate the result? This is really simple:

```
# render AdminState.dtl
$Output .= $Self->{LayoutObject}->Output (
    TemplateFile => 'AdminState',
    Data          => \%Param,
);
```

In the frontend modules, the `Output()` function of `Kernel::Output::HTML::Layout` is called (after all the needed blocks have been called in this template) to generate the final output. An optional set of data parameters is passed to the template, for all data inserting commands which are not inside of a block.

How to Extend it

Module Format

Core Modules

Agent Authentication Module

There are several agent authentication modules (DB, LDAP and HTTPBasicAuth) which come with the OTRS framework. It is also possible to develop your own authentication modules. The agent authentication modules are located under `Kernel/System/Auth/*.pm`. For more information about their configuration see the admin manual. Following, there is an example of a simple agent auth module. Save it under `Kernel/System/Auth/Simple.pm`. You just need 3 functions: `new()`, `GetOption()` and `Auth()`. Return the uid, then the authentication is ok.

Code Example

The interface class is called `Kernel::System::Auth`. The example agent authentication may be called `Kernel::System::Auth::CustomAuth`. You can find an example below.

```
# --
# Kernel/System/Auth/CustomAuth.pm - provides the CustomAuth
# authentication
# based on Martin Edenhofer's Kernel::System::Auth::DB
# Copyright (C) 2001-2010 OTRS AG, http://otrs.org/
# --
# ID: CustomAuth.pm,v 1.1 2010/05/10 15:30:34 fk Exp $
# --
```

```

# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::Auth::CustomAuth;

use strict;
use warnings;

use Authen::CustomAuth;

use vars qw($VERSION);
$VERSION = qw($Revision: 1.1 $) [1];

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # check needed objects
    for (qw(LogObject ConfigObject DBObject)) {
        $Self->{$_} = $Param{$_} || die "No $_!";
    }

    # Debug 0=off 1=on
    $Self->{Debug} = 0;

    # get config
    $Self->{Die} = $Self->{ConfigObject}-
>Get( 'AuthModule::CustomAuth::Die' . $Param{Count} );

    # get user table
    $Self->{CustomAuthHost} = $Self->{ConfigObject}-
>Get( 'AuthModule::CustomAuth::Host' . $Param{Count} )
        || die "Need AuthModule::CustomAuth::Host$Param{Count}.";
    $Self->{CustomAuthSecret}
        = $Self->{ConfigObject}-
>Get( 'AuthModule::CustomAuth::Password' . $Param{Count} )
        || die "Need AuthModule::CustomAuth::Password$Param{Count}.";

    return $Self;
}

sub GetOption {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    if ( !$Param{What} ) {
        $Self->{LogObject}->Log( Priority => 'error', Message => "Need
What!" );
    }
    return;
}

```

```
}

# module options
my %Option = ( PreAuth => 0, );

# return option
return $Option{ $Param{What} };
}

sub Auth {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    if ( !$Param{User} ) {
        $Self->{LogObject}->Log( Priority => 'error', Message => "Need
User!" );
        return;
    }

    # get params
    my $User      = $Param{User}      || '';
    my $Pw        = $Param{Pw}        || '';
    my $RemoteAddr = $ENV{REMOTE_ADDR} || 'Got no REMOTE_ADDR env!';
    my $UserID    = '';
    my $GetPw     = '';

    # just in case for debug!
    if ( $Self->{Debug} > 0 ) {
        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message => "User: '$User' tried to authenticate with Pw:
'$Pw' ($RemoteAddr)",
        );
    }

    # just a note
    if ( !$User ) {
        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message => "No User given!!! (REMOTE_ADDR: $RemoteAddr)",
        );
        return;
    }

    # just a note
    if ( !$Pw ) {
        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message => "User: $User authentication without Pw!!!
(REMOTE_ADDR: $RemoteAddr)",
        );
        return;
    }
}
```

```

# Create a radius object
my $CustomAuth = Authen::CustomAuth->new(
    Host    => $Self->{CustomAuthHost},
    Secret => $Self->{CustomAuthSecret},
);
if ( !$CustomAuth ) {
    if ( $Self->{Die} ) {
        die "Can't connect to $Self->{CustomAuthHost}: $@";
    }
    else {
        $Self->{LogObject}->Log(
            Priority => 'error',
            Message => "Can't connect to $Self->{CustomAuthHost}:
$@",
        );
        return;
    }
}
my $AuthResult = $CustomAuth->check_pwd( $User, $Pw );

# login note
if ( defined($AuthResult) && $AuthResult == 1 ) {
    $Self->{LogObject}->Log(
        Priority => 'notice',
        Message => "User: $User authentication ok (REMOTE_ADDR:
$RemoteAddr).",
    );
    return $User;
}

# just a note
else {
    $Self->{LogObject}->Log(
        Priority => 'notice',
        Message => "User: $User authentication with wrong Pw!!!
(REMOTE_ADDR: $RemoteAddr)"
    );
    return;
}
}
1;

```

Configuration Example

There is the need to activate your custom agent authenticate module. This can be done using the perl configuration below. It is not recommended to use the xml configuration because you can lock you out via the sysconfig.

```

$Self->{'AuthModule'} =
'Kernel::System::Auth::CustomAuth';

```

Use Case Example

Useful authentication implementation could be a soap backend.

Release Availability

Name	Release
DB	1.0
HTTPBasicAuth	1.2
LDAP	1.0
Radius	1.3

Authentication Synchronisation Module

There is a LDAP authentication synchronisation module which come with the OTRS framework. It is also possible to develop your own authentication modules. The authentication synchronisation modules are located under `Kernel/System/Auth/Sync/*.pm`. For more information about their configuration see the admin manual. Following, there is an example of an authentication synchronisation module. Save it under `Kernel/System/Auth/Sync/CustomAuthSync.pm`. You just need 2 functions: `new()` and `Sync()`. Return 1, then the synchronisation is ok.

Code Example

The interface class is called `Kernel::System::Auth`. The example agent authentication may be called `Kernel::System::Auth::Sync::CustomAuthSync`. You can find an example below.

```
# --
# Kernel/System/Auth/Sync/CustomAuthSync.pm - provides the
# CustomAuthSync
# Copyright (C) 2001-2010 OTRS AG, http://otrs.org/
# --
# Id: CustomAuthSync.pm,v 1.9 2010/03/25 14:42:45 martin Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::Auth::Sync::CustomAuthSync;

use strict;
use warnings;
use Net::LDAP;

use vars qw($VERSION);
$VERSION = qw($Revision: 1.1 $) [1];

sub new {
    my ( $Type, %Param ) = @_;
```

```

# allocate new hash for object
my $Self = {};
bless( $Self, $Type );

# check needed objects
for (qw(LogObject ConfigObject DBObject UserObject GroupObject
EncodeObject)) {
    $Self->{$_} = $Param{$_} || die "No $_!";
}

# Debug 0=off 1=on
$Self->{Debug} = 0;

...

return $Self;
}

sub Sync {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    for (qw(User)) {
        if ( !$Param{$_} ) {
            $Self->{LogObject}->Log( Priority => 'error', Message =>
"Need $_!" );
            return;
        }
    }
    ...
    return 1;
}

```

Configuration Example

There is the need to activate your custom synchronisation authenticate module. This can be done using the perl configuration below. It is not recommended to use the xml configuration because you can lock you out via the sysconfig.

```

$Self->{'AuthSyncModule'} =
'Kernel::System::Auth::Sync::LDAP';

```

Use Case Examples

Useful synchronisation implementation could be a soap or radius backend.

Release Availability

Name	Release
LDAP	2.4

Caveats and Warnings

Please note that the synchronisation was part of the authentication class `Kernel::System::Auth` before framework 2.4.

Customer Authentication Module

There are several customer authentication modules (DB, LDAP and HTTPBasicAuth) which come with the OTRS framework. It is also possible to develop your own authentication modules. The customer authentication modules are located under `Kernel/System/CustomerAuth/*.pm`. For more information about their configuration see the admin manual. Following, there is an example of a simple customer auth module. Save it under `Kernel/System/CustomerAuth/Simple.pm`. You just need 3 functions: `new()`, `GetOption()` and `Auth()`. Return the uid, then the authentication is ok.

Code Example

The interface class is called `Kernel::System::CustomerAuth`. The example customer authentication may be called `Kernel::System::CustomerAuth::CustomAuth`. You can find an example below.

```
# --
# Kernel/System/CustomerAuth/CustomAuth.pm - provides the custom
# Authentication
# based on Martin Edenhofer's Kernel::System::Auth::DB
# Copyright (C) 2001-2010 OTRS AG, http://otrs.org/
# --
# Id: CustomAuth.pm,v 1.11 2009/09/22 15:16:05 mb Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::CustomerAuth::CustomAuth;

use strict;
use warnings;

use Authen::CustomAuth;

use vars qw($VERSION);
$VERSION = qw($Revision: 1.1 $) [1];

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # check needed objects
    for (qw(LogObject ConfigObject DBObject)) {
        $Self->{$_} = $Param{$_} || die "No $_!";
    }
}
```

```

# Debug 0=off 1=on
$self->{Debug} = 0;

# get config
$self->{Die}
    = $self->{ConfigObject}-
>Get( 'Customer::AuthModule::CustomAuth::Die' . $Param{Count} );

# get user table
$self->{CustomAuthHost}
    = $self->{ConfigObject}-
>Get( 'Customer::AuthModule::CustomAuth::Host' . $Param{Count} )
    || die "Need Customer::AuthModule::CustomAuth::Host
$Param{Count} in Kernel/Config.pm";
$self->{CustomAuthSecret}
    = $self->{ConfigObject}-
>Get( 'Customer::AuthModule::CustomAuth::Password' . $Param{Count} )
    || die "Need Customer::AuthModule::CustomAuth::Password
$Param{Count} in Kernel/Config.pm";

return $self;
}

sub GetOption {
    my ( $self, %Param ) = @_;

    # check needed stuff
    if ( !$Param{What} ) {
        $self->{LogObject}->Log( Priority => 'error', Message => "Need
What!" );
        return;
    }

    # module options
    my %Option = ( PreAuth => 0, );

    # return option
    return $Option{ $Param{What} };
}

sub Auth {
    my ( $self, %Param ) = @_;

    # check needed stuff
    if ( !$Param{User} ) {
        $self->{LogObject}->Log( Priority => 'error', Message => "Need
User!" );
        return;
    }

    # get params
    my $User      = $Param{User}      || '';
    my $Pw        = $Param{Pw}        || '';

```

```

my $RemoteAddr = $ENV{REMOTE_ADDR} || 'Got no REMOTE_ADDR env!';
my $UserID     = '';
my $GetPw      = '';

# just in case for debug!
if ( $Self->{Debug} > 0 ) {
    $Self->{LogObject}->Log(
        Priority => 'notice',
        Message => "User: '$UserID' tried to authenticate with
Pw: '$Pw' ($RemoteAddr)",
    );
}

# just a note
if ( !$UserID ) {
    $Self->{LogObject}->Log(
        Priority => 'notice',
        Message => "No User given!!! (REMOTE_ADDR: $RemoteAddr)",
    );
    return;
}

# just a note
if ( !$Pw ) {
    $Self->{LogObject}->Log(
        Priority => 'notice',
        Message => "User: $UserID Authentication without Pw!!!
(REMOTE_ADDR: $RemoteAddr)",
    );
    return;
}

# Create a custom object
my $CustomAuth = Authen::CustomAuth->new(
    Host    => $Self->{CustomAuthHost},
    Secret => $Self->{CustomAuthSecret},
);
if ( !$CustomAuth ) {
    if ( $Self->{Die} ) {
        die "Can't connect to $Self->{CustomAuthHost}: $@";
    }
    else {
        $Self->{LogObject}->Log(
            Priority => 'error',
            Message => "Can't connect to $Self->{CustomAuthHost}:
$@",
        );
        return;
    }
}
my $AuthResult = $CustomAuth->check_pwd( $UserID, $Pw );

# login note
if ( defined($AuthResult) && $AuthResult == 1 ) {

```

```

        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message => "User: $User Authentication ok (REMOTE_ADDR:
$RemoteAddr).",
        );
        return $User;
    }

    # just a note
    else {
        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message => "User: $User Authentication with wrong Pw!!!
(REMOTE_ADDR: $RemoteAddr)"
        );
        return;
    }
}

1;

```

Configuration Example

There is the need to activate your custom customer authenticate module. This can be done using the xml configuration below.

```

<ConfigItem Name="AuthModule" Required="1" Valid="1">
    <Description Lang="en">Module to authenticate customers.</
Description>
    <Description Lang="de">Modul zum Authentifizieren der Customer.</
Description>
    <Group>Framework</Group>
    <SubGroup>Frontend::CustomerAuthAuth</SubGroup>
    <Setting>
        <Option Location="Kernel/System/CustomerAuth/*.pm"
SelectedID="Kernel::System::CustomerAuth::CustomAuth"></Option>
    </Setting>
</ConfigItem>

```

Use Case Example

Useful authentication implementation could be a soap backend.

Release Availability

Name	Release
DB	1.0
HTTPBasicAuth	1.2
LDAP	1.0
Radius	1.3

Customer User Preferences Module

There is a DB customer-user preferences module which come with the OTRS framework. It is also possible to develop your own customer-user preferences modules. The customer-user preferences modules are located under `Kernel/System/CustomerUser/Preferences/*.pm`. For more information about their configuration see the admin manual. There is an example of a customer-user preferences module below. Save it under `Kernel/System/CustomerUser/Preferences/Custom.pm`. You just need 4 functions: `new()`, `SearchPreferences()`, `SetPreferences()` and `GetPreferences()`.

Code Example

The interface class is called `Kernel::System::CustomerUser`. The example customer-user preferences may be called `Kernel::System::CustomerUser::Preferences::Custom`. You can find an example below.

```
# --
# Kernel/System/CustomerUser/Preferences/Custom.pm - some customer
# user functions
# Copyright (C) 2001-2010 OTRS AG, http://otrs.org/
# --
# Id: Custom.pm,v 1.20 2009/10/07 20:41:50 martin Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::CustomerUser::Preferences::Custom;

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.1 $) [1];

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # check needed objects
    for my $Object (qw(DBObject ConfigObject LogObject)) {
        $Self->{$Object} = $Param{$Object} || die "Got no $Object!";
    }

    # preferences table data
    $Self->{PreferencesTable} = $Self->{ConfigObject}-
>Get('CustomerPreferences')->{Params}->{Table}
        || 'customer_preferences';
    $Self->{PreferencesTableKey}
```

```

        = $Self->{ConfigObject}->Get('CustomerPreferences')->{Params}-
>{TableKey}
        || 'preferences_key';
        $Self->{PreferencesTableValue}
        = $Self->{ConfigObject}->Get('CustomerPreferences')->{Params}-
>{TableValue}
        || 'preferences_value';
        $Self->{PreferencesTableUserID}
        = $Self->{ConfigObject}->Get('CustomerPreferences')->{Params}-
>{TableUserID}
        || 'user_id';

    return $Self;
}

sub SetPreferences {
    my ( $Self, %Param ) = @_;

    my $UserID = $Param{UserID} || return;
    my $Key     = $Param{Key}     || return;
    my $Value = defined( $Param{Value} ) ? $Param{Value} : '';

    # delete old data
    return if !$Self->{DBObject}->Do(
        SQL => "DELETE FROM $Self->{PreferencesTable} WHERE "
            . " $Self->{PreferencesTableUserID} = ? AND $Self-
>{PreferencesTableKey} = ?",
        Bind => [ \$UserID, \$Key ],
    );

    $Value .= 'Custom';

    # insert new data
    return if !$Self->{DBObject}->Do(
        SQL => "INSERT INTO $Self->{PreferencesTable} ($Self-
>{PreferencesTableUserID}, "
            . " $Self->{PreferencesTableKey}, $Self-
>{PreferencesTableValue}) "
            . " VALUES (?, ?, ?)",
        Bind => [ \$UserID, \$Key, \$Value ],
    );

    return 1;
}

sub GetPreferences {
    my ( $Self, %Param ) = @_;

    my $UserID = $Param{UserID} || return;
    my %Data;

    # get preferences

    return if !$Self->{DBObject}->Prepare(

```

```

        SQL => "SELECT $Self->{PreferencesTableKey}, $Self-
>{PreferencesTableValue} "
        . " FROM $Self->{PreferencesTable} WHERE $Self-
>{PreferencesTableUserID} = ?",
        Bind => [ \$UserID ],
    );
    while ( my @Row = $Self->{DBObject}->FetchrowArray() ) {
        $Data{ $Row[0] } = $Row[1];
    }

    # return data
    return %Data;
}

sub SearchPreferences {
    my ( $Self, %Param ) = @_;

    my %UserID;
    my $Key   = $Param{Key}   || '';
    my $Value = $Param{Value} || '';

    # get preferences
    my $SQL = "SELECT $Self->{PreferencesTableUserID}, $Self-
>{PreferencesTableValue} "
        . " FROM "
        . " $Self->{PreferencesTable} "
        . " WHERE "
        . " $Self->{PreferencesTableKey} = '"
        . $Self->{DBObject}->Quote($Key) . "'" . " AND "
        . " LOWER($Self->{PreferencesTableValue}) LIKE LOWER('"
        . $Self->{DBObject}->Quote( $Value, 'Like' ) . "'" );

    return if !$Self->{DBObject}->Prepare( SQL => $SQL );
    while ( my @Row = $Self->{DBObject}->FetchrowArray() ) {
        $UserID{ $Row[0] } = $Row[1];
    }

    # return data
    return %UserID;
}

1;

```

Configuration Example

There is the need to activate your custom customer-user preferences module. This can be done using the xml configuration below.

```

<ConfigItem Name="CustomerPreferences" Required="1" Valid="1">
    <Description Lang="en">Parameters for the customer preference
table.</Description>

```

```

    <Description Lang="de">Parameter für die Tabelle mit den
Einstellungen für die Customer.</Description>
    <Group>Framework</Group>
    <SubGroup>Frontend::Customer::Preferences</SubGroup>
    <Setting>
        <Hash>
            <Item
Key="Module">Kernel::System::CustomerUser::Preferences::Custom</Item>
            <Item Key="Params">
                <Hash>
                    <Item Key="Table">customer_preferences</Item>
                    <Item Key="TableKey">preferences_key</Item>
                    <Item Key="TableValue">preferences_value</Item>
                    <Item Key="TableUserID">user_id</Item>
                </Hash>
            </Item>
        </Hash>
    </Setting>
</ConfigItem>

```

Use Case Example

Useful preferences implementation could be a soap or ldap backend.

Release Availability

Name	Release
DB	2.3

Log Module

There is a global log interface for OTRS that provides the possibility to create own log backends.

Writing an own logging backend is as easy as reimplementing the `Kernel::System::Log::Log()` method.

Code example: `Kernel::System::Log::CustomFile`

In this small example, we'll write a little file logging backend which works similar to `Kernel::System::Log::File`, but prepends a string to each logging entry.

```

# --
# Kernel/System/Log/CustomFile.pm - file log backend
# Copyright (C) 2001-2010 OTRS AG, http://otrs.org/
# --
# $Id: log.xml,v 1.1 2010/08/13 08:59:28 mg Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

```

```
package Kernel::System::Log::CustomFile;

use strict;
use warnings;

use vars qw($VERSION);
$VERSION = qw($Revision: 1.1 $) [1];

umask "002";

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # get needed objects
    for (qw(ConfigObject EncodeObject)) {
        if ( $Param{$_} ) {
            $Self->{$_} = $Param{$_};
        }
        else {
            die "Got no $_!";
        }
    }

    # get logfile location
    $Self->{LogFile} = '/var/log/CustomFile.log';

    # set custom prefix
    $Self->{CustomPrefix} = 'CustomFileExample';

    # Fixed bug# 2265 - For IIS we need to create a own error log
    file.
    # Bind stderr to log file, because iis do print stderr to web
    page.
    if ( $ENV{SERVER_SOFTWARE} && $ENV{SERVER_SOFTWARE} =~ /^microsoft
    \-iis/i ) {
        if ( !open STDERR, '>>', $Self->{LogFile} . '.error' ) {
            print STDERR "ERROR: Can't write $Self->{LogFile}.error:
            $!";
        }
    }

    return $Self;
}

sub Log {
    my ( $Self, %Param ) = @_;

    my $FH;

    # open logfile
```

```

if ( !open $FH, '>>', $Self->{LogFile} ) {

    # print error screen
    print STDERR "\n";
    print STDERR " >> Can't write $Self->{LogFile}: $! <<\n";
    print STDERR "\n";
    return;
}

# write log file
$Self->{EncodeObject}->SetIO($FH);
print $FH '[' . localtime() . ']';
if ( lc $Param{Priority} eq 'debug' ) {
    print $FH "[Debug][$Param{Module}][$Param{Line}] $Self->{CustomPrefix} $Param{Message}\n";
}
elseif ( lc $Param{Priority} eq 'info' ) {
    print $FH "[Info][$Param{Module}] $Self->{CustomPrefix} $Param{Message}\n";
}
elseif ( lc $Param{Priority} eq 'notice' ) {
    print $FH "[Notice][$Param{Module}] $Self->{CustomPrefix} $Param{Message}\n";
}
elseif ( lc $Param{Priority} eq 'error' ) {
    print $FH "[Error][$Param{Module}][$Param{Line}] $Self->{CustomPrefix} $Param{Message}\n";
}
else {

    # print error messages to STDERR
    print STDERR
        "[Error][$Param{Module}] $Self->{CustomPrefix} Priority: '$Param{Priority}' not defined! Message: $Param{Message}\n";

    # and of course to logfile
    print $FH
        "[Error][$Param{Module}] $Self->{CustomPrefix} Priority: '$Param{Priority}' not defined! Message: $Param{Message}\n";
}

# close file handle
close $FH;
return 1;
}

1;

```

Configuration example

To activate our custom logging module, the administrator can either set the existing configuration item "LogModule" manually to "Kernel::System::Log::CustomFile". To realize this automatically, you can provide an XML configuration file which overrides the default setting.

```
<ConfigItem Name="LogModule" Required="1" Valid="1">
  <Description Translatable="1">Set Kernel::System::Log::CustomFile
  as default logging backend.</Description>
  <Group>Framework</Group>
  <SubGroup>Core::Log</SubGroup>
  <Setting>
    <Option Location="Kernel/System/Log/*.pm"
    SelectedID="Kernel::System::Log::CustomFile"></Option>
  </Setting>
</ConfigItem>
```

Use case examples

Useful logging backends could be logging to a web service or to encrypted files.

Caveats and Warnings

Please note that Kernel::System::Log has other methods than Log() which cannot be reimplemented, for example code for working with shared memory segments and log data caching.

Output Filter

Output filters allow to modify HTML on the fly. It is best practice to use output filters instead of modifying `.dtl` files directly. There are three good reasons for that. When the same adaption has to be applied to several frontend modules then the adaption only has to be implemented once. The second advantage is that when OTRS is upgraded there is a chance that the filter doesn't have to be updated, when the relevant pattern has not changed. When two extensions modify the same file there is a conflict during the installation of the second package. This conflict can be resolved by using two output filters that modify the same frontend module.

There are four different kinds of output filters. They are active at different stages of the generation of HTML content.

FilterElementPre

The content of a template can be changed by the filter before any processing by the Layout module takes place. This kind of filter should be used in most cases. Processing instructions like `$Text{"..."}`, `$QData{"..."}` can be inserted into the template content and they will be honored by the subsequent DTL processing.

FilterElementPost

The content of a template can be changed after variable substitution and translation. The kind of filter should only be used when the filter needs access to translated strings or to substituted variables.

FilterContent

This kind of filter allows to process the complete HTML output for the request right before it is sent to the browser. This can be used for global transformations. But in real live there is rarely a need to use this kind of filter.

FilterText

This kind of output filter is a plugin for the method `Kernel::Output::HTML::Layout::Ascii2HTML()` and is only active when the parameter `LinkFeature` is set to 1. Thus the `FilterText` output filters are currently only active for the display of the body of plain text articles. Plain text articles are generated by incoming non-HTML mails and when OTRS is configured to not use the rich text feature in the frontend.

Code example

See package `TemplateModule`.

Configuration example

See package `TemplateModule`.

Use Cases

Show additional ticket attributes in `AgentTicketZoom`.

All ticket attributes are passed to the `AgentTicketZoom` template. Therefore it suffices to insert e.g. the instruction `$QData{"Title"}` into the content. This can be achieved with a `FilterElementPre` output filter.

Add an additional CSS file.

An additional CSS file can be added to all agent frontends with an `FilterElementPre` filter that only modifies `Header.dtl`. Therefore it suffices to insert e.g. the instruction `$QData{"Title"}` into the content. This can be achieved with a `FilterElementPre` output filter.

Show the service selection as a multi level menu.

Use a `FilterElementPost` for this feature. The list of selectable services can be parsed from the processed template output. The multi level selection can be constructed from the service list and inserted into the template content. A `FilterElementPost` output filter must be used for that.

Create links within plain text article bodies.

A biotech company uses gene names like `IPI00217472` in plain text articles. A `FilterText` output filter can be used to create links to a sequence database, e.g. `http://srs.ebi.ac.uk/srsbin/cgi-bin/wgetz?-e+[IPI-acc:IPI00217472]+-vn+2`, for the gene names.

Prohibit active content

There is firewall rule that disallows all active content. In order to avoid rejection by the firewall the HTML tag `<applet>` can be filtered with an `FilterContent` output filter.

Caveats and Warnings

Every `ElementPre` and `ElementPost` output filter is constructed and run for every `Template` that is needed for the current request. Thus low performance of the output filter or a large number of filters can severely degrade performance. When that becomes an issue, the construction of needed objects can be done in the `Run`-method after the checks. Thus the expensive code is run only in the relevant cases.

Best Practices

In order to increase flexibility the list of affected templates should be configurable in SysConfig.

Release Availability

The four kinds of output filters are available in OTRS 2.4.

Queue Preferences Module

There is a DB queue preferences module which come with the OTRS framework. It is also possible to develop your own queue preferences modules. The queue preferences modules are located under Kernel/System/Queue/*.pm. For more information about their configuration see the admin manual. Following, there is an example of a queue preferences module. Save it under Kernel/System/Queue/PreferencesCustom.pm. You just need 3 functions: new(), QueuePreferencesSet() and QueuePreferencesGet(). Return 1, then the synchronisation is ok.

Code Example

The interface class is called Kernel::System::Queue. The example queue preferences may be called Kernel::System::Queue::PreferencesCustom. You can find an example below.

```
# --
# Kernel/System/Queue/PreferencesCustom.pm - some user functions
# Copyright (C) 2001-2010 OTRS AG, http://otrs.org/
# --
# Id: PreferencesCustom.pm,v 1.5 2009/02/16 11:47:34 tr Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::Queue::PreferencesCustom;

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.1 $) [1];

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # check needed objects
    for (qw(DBObject ConfigObject LogObject)) {
        $Self->{$_} = $Param{$_} || die "Got no $_!";
    }
}
```

```

# preferences table data
$Self->{PreferencesTable}      = 'queue_preferences';
$Self->{PreferencesTableKey}    = 'preferences_key';
$Self->{PreferencesTableValue}  = 'preferences_value';
$Self->{PreferencesTableQueueID} = 'queue_id';

return $Self;
}

sub QueuePreferencesSet {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    for (qw(QueueID Key Value)) {
        if ( !defined( $Param{$_} ) ) {
            $Self->{LogObject}->Log( Priority => 'error', Message =>
"Need $_!" );
            return;
        }
    }

    # delete old data
    return if !$Self->{DBObject}->Do(
        SQL => "DELETE FROM $Self->{PreferencesTable} WHERE "
            . "$Self->{PreferencesTableQueueID} = ? AND $Self->{PreferencesTableKey} = ?",
        Bind => [ \ $Param{QueueID}, \ $Param{Key} ],
    );

    $Self->{PreferencesTableValue} .= 'PreferencesCustom';

    # insert new data
    return $Self->{DBObject}->Do(
        SQL => "INSERT INTO $Self->{PreferencesTable} ($Self->{PreferencesTableQueueID}, "
            . " $Self->{PreferencesTableKey}, $Self->{PreferencesTableValue}) "
            . " VALUES (?, ?, ?)",
        Bind => [ \ $Param{QueueID}, \ $Param{Key}, \ $Param{Value} ],
    );
}

sub QueuePreferencesGet {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    for (qw(QueueID)) {
        if ( !$Param{$_} ) {
            $Self->{LogObject}->Log( Priority => 'error', Message =>
"Need $_!" );
            return;
        }
    }
}

```

```

# check if queue preferences are available
if ( !$Self->{ConfigObject}->Get('QueuePreferences') ) {
    return;
}

# get preferences
return if !$Self->{DBObject}->Prepare(
    SQL => "SELECT $Self->{PreferencesTableKey}, $Self-
>{PreferencesTableValue} "
    . " FROM $Self->{PreferencesTable} WHERE $Self-
>{PreferencesTableQueueID} = ?",
    Bind => [ \ $Param{QueueID} ],
);
my %Data;
while ( my @Row = $Self->{DBObject}->FetchrowArray() ) {
    $Data{ $Row[0] } = $Row[1];
}

# return data
return %Data;
}

1;

```

Configuration Example

There is the need to activate your custom queue preferences module. This can be done using the xml configuration below.

```

<ConfigItem Name="Queue::PreferencesModule" Required="1" Valid="1">
    <Description Lang="en">Default queue preferences module.</
Description>
    <Description Lang="de">Standard Queue Preferences Module.</
Description>
    <Group>Ticket</Group>
    <SubGroup>Frontend::Queue::Preferences</SubGroup>
    <Setting>
        <String Regex="">Kernel::System::Queue::PreferencesCustom</
String>
    </Setting>
</ConfigItem>

```

Use Case Examples

Useful preferences implementation could be a soap or radius backend.

Release Availability

Name	Release
PreferencesDB	2.3

Service Preferences Module

There is a DB service preferences module which come with the OTRS framework. It is also possible to develop your own service preferences modules. The service preferences modules are located under Kernel/System/Service/*.pm. For more information about their configuration see the admin manual. Following, there is an example of a service preferences module. Save it under Kernel/System/Service/PreferencesCustom.pm. You just need 3 functions: new(), ServicePreferencesSet() and ServicePreferencesGet(). Return 1, then the synchronisation is ok.

Code Example

The interface class is called Kernel::System::Service. The example service preferences may be called Kernel::System::Service::PreferencesCustom. You can find an example below.

```
# --
# Kernel/System/Service/PreferencesCustom - some user functions
# Copyright (C) 2001-2010 OTRS AG, http://otrs.org/
# --
# Id: PreferencesCustom.pm,v 1.2 2009/02/16 11:47:34 tr Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::Service::PreferencesCustom;

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.1 $) [1];

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # check needed objects
    for (qw(DBObject ConfigObject LogObject)) {
        $Self->{$_} = $Param{$_} || die "Got no $_!";
    }

    # preferences table data
    $Self->{PreferencesTable}           = 'service_preferences';
    $Self->{PreferencesTableKey}        = 'preferences_key';
    $Self->{PreferencesTableValue}     = 'preferences_value';
    $Self->{PreferencesTableServiceID} = 'service_id';

    return $Self;
}
```

```

sub ServicePreferencesSet {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    for (qw(ServiceID Key Value)) {
        if ( !defined( $Param{$_} ) ) {
            $Self->{LogObject}->Log( Priority => 'error', Message =>
"Need $_!" );
            return;
        }
    }

    # delete old data
    return if !$Self->{DBObject}->Do(
        SQL => "DELETE FROM $Self->{PreferencesTable} WHERE "
            . "$Self->{PreferencesTableServiceID} = ? AND $Self-
>{PreferencesTableKey} = ?",
        Bind => [ \ $Param{ServiceID}, \ $Param{Key} ],
    );

    $Self->{PreferencesTableValue} .= 'PreferencesCustom';

    # insert new data
    return $Self->{DBObject}->Do(
        SQL => "INSERT INTO $Self->{PreferencesTable} ($Self-
>{PreferencesTableServiceID}, "
            . " $Self->{PreferencesTableKey}, $Self-
>{PreferencesTableValue}) "
            . " VALUES (?, ?, ?)",
        Bind => [ \ $Param{ServiceID}, \ $Param{Key}, \ $Param{Value} ],
    );
}

sub ServicePreferencesGet {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    for (qw(ServiceID)) {
        if ( !$Param{$_} ) {
            $Self->{LogObject}->Log( Priority => 'error', Message =>
"Need $_!" );
            return;
        }
    }

    # check if service preferences are available
    if ( !$Self->{ConfigObject}->Get('ServicePreferences') ) {
        return;
    }

    # get preferences
    return if !$Self->{DBObject}->Prepare(
        SQL => "SELECT $Self->{PreferencesTableKey}, $Self-
>{PreferencesTableValue} "

```

```

        . " FROM $Self->{PreferencesTable} WHERE $Self-
>{PreferencesTableServiceID} = ?",
        Bind => [ \$Param{ServiceID} ],
    );
    my %Data;
    while ( my @Row = $Self->{DBObject}->FetchrowArray() ) {
        $Data{ $Row[0] } = $Row[1];
    }

    # return data
    return %Data;
}

1;

```

Configuration Example

There is the need to activate your custom service preferences module. This can be done using the xml configuration below.

```

<ConfigItem Name="Service::PreferencesModule" Required="1" Valid="1">
    <Description Lang="en">Default service preferences module.</
Description>
    <Description Lang="de">Standard Service Preferences Module.</
Description>
    <Group>Ticket</Group>
    <SubGroup>Frontend::Service::Preferences</SubGroup>
    <Setting>
        <String Regex="">Kernel::System::Service::PreferencesCustom</
String>
    </Setting>
</ConfigItem>

```

Use Case Example

Useful preferences implementation could be a soap or radius backend.

Release Availability

Name	Release
PreferencesDB	2.4

SLA Preferences Module

There is a DB sla preferences module which come with the OTRS framework. It is also possible to develop your own sla preferences modules. The sla preferences modules are located under Kernel/System/SLA/*.pm. For more information about their configuration see the admin manual. Following, there is an example of a sla preferences module. Save it under Kernel/System/SLA/PreferencesCustom.pm. You just need 3 functions: new(), SLAPreferencesSet() and SLAPreferencesGet(). Return 1, then the synchronisation is ok.

Code Example

The interface class is called `Kernel::System::SLA`. The example `sla` preferences may be called `Kernel::System::SLA::PreferencesCustom`. You can find an example below.

```
# --
# Kernel/System/SLA/PreferencesCustom.pm - some user functions
# Copyright (C) 2001-2010 OTRS AG, http://otrs.org/
# --
# Id: PreferencesCustom.pm,v 1.2 2009/02/16 11:47:34 tr Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::SLA::PreferencesCustom;

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.1 $) [1];

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # check needed objects
    for (qw(DBObject ConfigObject LogObject)) {
        $Self->{$_} = $Param{$_} || die "Got no $_!";
    }

    # preferences table data
    $Self->{PreferencesTable}      = 'sla_preferences';
    $Self->{PreferencesTableKey}   = 'preferences_key';
    $Self->{PreferencesTableValue} = 'preferences_value';
    $Self->{PreferencesTableSLAID} = 'sla_id';

    return $Self;
}

sub SLAPreferencesSet {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    for (qw(SLAID Key Value)) {
        if ( !defined( $Param{$_} ) ) {
            $Self->{LogObject}->Log( Priority => 'error', Message =>
                "Need $_!" );
        }
    }
}

```

```

        return;
    }
}

# delete old data
return if !$Self->{DBObject}->Do(
    SQL => "DELETE FROM $Self->{PreferencesTable} WHERE "
        . "$Self->{PreferencesTableSLAID} = ? AND $Self-
>{PreferencesTableKey} = ?",
    Bind => [ \ $Param{SLAID}, \ $Param{Key} ],
);

$Self->{PreferencesTableValue} .= 'PreferencesCustom';

# insert new data
return $Self->{DBObject}->Do(
    SQL => "INSERT INTO $Self->{PreferencesTable} ($Self-
>{PreferencesTableSLAID}, "
        . " $Self->{PreferencesTableKey}, $Self-
>{PreferencesTableValue}) "
        . " VALUES (?, ?, ?)",
    Bind => [ \ $Param{SLAID}, \ $Param{Key}, \ $Param{Value} ],
);
}

sub SLAPreferencesGet {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    for (qw(SLAID)) {
        if ( !$Param{$_} ) {
            $Self->{LogObject}->Log( Priority => 'error', Message =>
"Need $_!" );
            return;
        }
    }

    # check if service preferences are available
    if ( !$Self->{ConfigObject}->Get('SLAPreferences') ) {
        return;
    }

    # get preferences
    return if !$Self->{DBObject}->Prepare(
        SQL => "SELECT $Self->{PreferencesTableKey}, $Self-
>{PreferencesTableValue} "
            . " FROM $Self->{PreferencesTable} WHERE $Self-
>{PreferencesTableSLAID} = ?",
        Bind => [ \ $Param{SLAID} ],
    );
    my %Data;
    while ( my @Row = $Self->{DBObject}->FetchrowArray() ) {
        $Data{ $Row[0] } = $Row[1];
    }
}

```

```

    # return data
    return %Data;
}

1;

```

Configuration Example

There is the need to activate your custom sla preferences module. This can be done using the xml configuration below.

```

<ConfigItem Name="SLA::PreferencesModule" Required="1" Valid="1">
  <Description Lang="en">Default sla preferences module.</
Description>
  <Description Lang="de">Standard SLA Preferences Module.</
Description>
  <Group>Ticket</Group>
  <SubGroup>Frontend::SLA::Preferences</SubGroup>
  <Setting>
    <String Regex="">Kernel::System::SLA::PreferencesCustom</
String>
  </Setting>
</ConfigItem>

```

Use Case Example

Useful preferences implementation could be a soap or radius backend.

Release Availability

Name	Release
PreferencesDB	2.4

Stats Module

There are two different types of internal stats modules - dynamic and static. This section describes how such stats modules can be developed.

Dynamic Stats

In contrast to static stats modules, dynamic statistics can be configured via the OTRS web interface. In this section a simple statistic module is developed. Each dynamic stats module has to implement these subroutines

- new
- GetObjectName
- GetObjectAttributes

- ExportWrapper
- ImportWrapper

Furthermore the module has to implement either `GetStatElement` or `GetStatTable`. And if the header line of the result table should be changed, a sub called `GetHeaderLine` has to be developed.

Code example

In this section a sample stats module is shown and each subroutine is explained.

```
# --
# Kernel/System/Stats/Dynamic/DynamicStatsTemplate.pm - all advice
# functions
# Copyright (C) 2001-2010 OTRS AG, http://otrs.org/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::Stats::Dynamic::DynamicStatsTemplate;

use strict;
use warnings;

use Kernel::System::Queue;
use Kernel::System::State;
use Kernel::System::Ticket;

use vars qw($VERSION);
$VERSION = qw($Revision: 1.1 $) [1];
```

This is common boilerplate that can be found in common OTRS modules. The class/package name is declared via the `package` keyword. Then the needed modules are 'use'd.

```
sub new {
    my ( $Type, %Param ) = @_ ;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # check needed objects
    for my $Object (
        qw(DBObject ConfigObject LogObject UserObject TimeObject
MainObject EncodeObject)
    )
    {
        $Self->{$Object} = $Param{$Object} || die "Got no $Object!";
    }
}
```

```

# created needed objects
$self->{QueueObject} = Kernel::System::Queue->new( %{$self} );
$self->{TicketObject} = Kernel::System::Ticket->new( %{$self} );
$self->{StateObject} = Kernel::System::State->new( %{$self} );

return $self;
}

```

`new` is the constructor for this statistic module. It creates a new instance of the class. According to the coding guidelines objects of other classes that are needed in this module have to be created in "new". In lines 27 to 29 the object of the stats module is created. Lines 31 to 37 check if objects that are needed in this code - either for creating other objects or in this module - are passed. After that the other objects are created.

```

sub GetObjectName {
    my ( $self, %param ) = @_;

    return 'Sample Statistics';
}

```

`GetObjectName` returns a Name for the Statistics module. This is the label that is shown in the drop down in the configuration as well as in the list of existing statistics (column "object").

```

sub GetObjectAttributes {
    my ( $self, %param ) = @_;

    # get state list
    my %stateList = $self->{StateObject}->StateList(
        UserID => 1,
    );

    # get queue list
    my %queueList = $self->{QueueObject}->GetAllQueues();

    # get current time to fix bug#3830
    my $timestamp = $self->{TimeObject}->CurrentTimestamp();
    my ($date) = split /\s+/, $timestamp;
    my $today = sprintf "%s 23:59:59", $date;

    my @objectAttributes = (
        {
            Name           => 'State',
            UseAsXvalue    => 1,
            UseAsValueSeries => 1,
            UseAsRestriction => 1,
            Element        => 'StateIDs',
            Block          => 'MultiSelectField',
            Values         => \%stateList,
        },
    );
}

```

```

        {
            Name           => 'Created in Queue',
            UseAsXvalue    => 1,
            UseAsValueSeries => 1,
            UseAsRestriction => 1,
            Element        => 'CreatedQueueIDs',
            Block          => 'MultiSelectField',
            Translation     => 0,
            Values          => \%QueueList,
        },
        {
            Name           => 'Create Time',
            UseAsXvalue    => 1,
            UseAsValueSeries => 1,
            UseAsRestriction => 1,
            Element        => 'CreateTime',
            TimePeriodFormat => 'DateInputFormat',      #
'DateInputFormatLong',
            Block          => 'Time',
            TimeStop       => $Today,
            Values          => {
                TimeStart => 'TicketCreateTimeNewerDate',
                TimeStop  => 'TicketCreateTimeOlderDate',
            },
        },
    },
);

return @ObjectAttributes;
}

```

In this sample stats module, we want to provide three attributes the user can chose from: A list of queues, a list of states an a time drop down. To get the values shown in the drop down, some operations are needed. In this case call StateList and GetAllQueues.

Then the list of attributes is created. Each attribute is defined via a hashreference. You can use these keys:

- Name
the label in the web interface
- UseAsXvalue
Can this attribute be used on the x-axis
- UseAsValueSeries
Can this attribute be used on the y-axis
- UseAsRestriction
Can this attribute be used for restrictions.
- Element

the HTML fieldname

- Block

the block name in the template file (e.g. <OTRS_HOME>/Kernel/Output/HTML/Standard/AgentStatsEditXaxis.dtl)

- Values

the values shown in the attribute

Hint: If you install this sample and you configured a statistic with some queues - lets say 'queue A' and 'queue B' - then these queues are the only ones that are shown to the user when he starts the statistic. Sometimes a dynamic drop down or multiselect field is needed. In this case, you can set "SelectedValues" in the definition of the attribute:

```
{
    Name           => 'Created in Queue',
    UseAsXvalue    => 1,
    UseAsValueSeries => 1,
    UseAsRestriction => 1,
    Element        => 'CreatedQueueIDs',
    Block          => 'MultiSelectField',
    Translation    => 0,
    Values         => \%QueueList,
    SelectedValues => [ @SelectedQueues ],
},
```

```
sub GetStatElement {
    my ( $Self, %Param ) = @_;

    # search tickets
    return $Self->{TicketObject}->TicketSearch(
        UserID      => 1,
        Result      => 'COUNT',
        Permission  => 'ro',
        Limit       => 100_000_000,
        %Param,
    );
}
```

GetStatElement gets called for each cell in the result table. So it should be a numeric value. In this sample it does a simple ticket search. The hash %Param contains information about the "current" x-value and the y-value as well as any restrictions. So, for a cell that should count the created tickets for queue 'Misc' with state 'open' the passed parameter hash looks something like this:

```
'CreatedQueueIDs' => [
    '4'
],
```

```
'StateIDs' => [
    '2'
]
```

If the "per cell" calculation should be avoided, GetStatTable is an alternative. GetStatTable returns a list of rows, hence an array of arrayreferences. This leads to the same result as using GetStatElement

```
sub GetStatTable {
    my ( $Self, %Param ) = @_;

    my @StatData;

    for my $StateName ( keys %{ $Param{TableStructure} } ) {
        my @Row;
        for my $Params ( @{ $Param{TableStructure}->{$StateName} } ) {
            my $Tickets = $Self->{TicketObject}->TicketSearch(
                UserID      => 1,
                Result      => 'COUNT',
                Permission => 'ro',
                Limit       => 100_000_000,
                %{$Params},
            );

            push @Row, $Tickets;
        }

        push @StatData, [ $StateName, @Row ];
    }

    return @StatData;
}
```

GetStatTable gets all information about the stats query that is needed. The passed parameters contains information about the attributes (Restrictions, attributes that are used for x/y-axis) and the table structure. The table structure is a hash reference where the keys are the values of the y-axis and their values are hashreferences with the parameters used for GetStatElement subroutines.

```
'Restrictions' => {},
'TableStructure' => {
    'closed successful' => [
        {
            'CreatedQueueIDs' => [
                '3'
            ],
            'StateIDs' => [
                '2'
            ]
        }
    ],
},
```

```

],
'closed unsuccessful' => [
  {
    'CreatedQueueIDs' => [
      '3'
    ],
    'StateIDs' => [
      '3'
    ]
  },
],
},
'ValueSeries' => [
  {
    'Block' => 'MultiSelectField',
    'Element' => 'StateIDs',
    'Name' => 'State',
    'SelectedValues' => [
      '5',
      '3',
      '2',
      '1',
      '4'
    ],
    'Translation' => 1,
    'Values' => {
      '1' => 'new',
      '10' => 'closed with workaround',
      '2' => 'closed successful',
      '3' => 'closed unsuccessful',
      '4' => 'open',
      '5' => 'removed',
      '6' => 'pending reminder',
      '7' => 'pending auto close+',
      '8' => 'pending auto close-',
      '9' => 'merged'
    }
  }
],
'XValue' => {
  'Block' => 'MultiSelectField',
  'Element' => 'CreatedQueueIDs',
  'Name' => 'Created in Queue',
  'SelectedValues' => [
    '3',
    '4',
    '1',
    '2'
  ],
  'Translation' => 0,
  'Values' => {
    '1' => 'Postmaster',
    '2' => 'Raw',
    '3' => 'Junk',

```

```

        '4' => 'Misc'
    }
}

```

Sometimes the headers of the table have to be changed. In that case, a subroutine called `GetHeaderLine` has to be implemented. That subroutine has to return an arrayreference with the column headers as elements. It gets information about the x-values passed.

```

sub GetHeaderLine {
    my ( $Self, %Param ) = @_;

    my @HeaderLine = ( '' );
    for my $SelectedXValue ( @ { $Param{XValue}->{SelectedValues} } ) {
        push @HeaderLine, $Param{XValue}->{Values}->{$SelectedXValue};
    }

    return \@HeaderLine;
}

```

```

sub ExportWrapper {
    my ( $Self, %Param ) = @_;

    # wrap ids to used spelling
    for my $Use (qw(UseAsValueSeries UseAsRestriction UseAsXvalue)) {
        ELEMENT:
        for my $Element ( @ { $Param{$Use} } ) {
            next ELEMENT if !$Element || !$Element->{SelectedValues};
            my $ElementName = $Element->{Element};
            my $Values      = $Element->{SelectedValues};

            if ( $ElementName eq 'QueueIDs' || $ElementName eq
                'CreatedQueueIDs' ) {
                ID:
                for my $ID ( @ { $Values } ) {
                    next ID if !$ID;
                    $ID->{Content} = $Self->{QueueObject}-
>QueueLookup( QueueID => $ID->{Content} );
                }
            }
            elsif ( $ElementName eq 'StateIDs' || $ElementName eq
                'CreatedStateIDs' ) {
                my %StateList = $Self->{StateObject}-
>StateList( UserID => 1 );
                ID:
                for my $ID ( @ { $Values } ) {
                    next ID if !$ID;
                    $ID->{Content} = $StateList{ $ID->{Content} };
                }
            }
        }
    }
}

```

```

    }
    return \%Param;
}

```

Configured statistics can be exported into XML format. But as queues with the same queue names can have different IDs on different OTRS instances it would be quite painful to export the IDs (the statistics would calculate the wrong numbers then). So an export wrapper should be written to use the names instead of ids. This should be done for each "dimension" of the stats module (x-axis, y-axis and restrictions).

ImportWrapper works the other way around - it converts the name to the ID in the instance the configuration is imported to.

This is a sample export:

```

<?xml version="1.0" encoding="utf-8"?>

<otrs_stats>
<Cache>0</Cache>
<Description>Sample stats module</Description>
<File></File>
<Format>CSV</Format>
<Format>Print</Format>
<Object>DeveloperManualSample</Object>
<ObjectModule>Kernel::System::Stats::Dynamic::DynamicStatsTemplate</ObjectModule>
<ObjectName>Sample Statistics</ObjectName>
<Permission>stats</Permission>
<StatType>dynamic</StatType>
<SumCol>0</SumCol>
<SumRow>0</SumRow>
<Title>Sample 1</Title>
<UseAsValueSeries Element="StateIDs" Fixed="1">
<SelectedValues>removed</SelectedValues>
<SelectedValues>closed unsuccessful</SelectedValues>
<SelectedValues>closed successful</SelectedValues>
<SelectedValues>new</SelectedValues>
<SelectedValues>open</SelectedValues>
</UseAsValueSeries>
<UseAsXvalue Element="CreatedQueueIDs" Fixed="1">
<SelectedValues>Junk</SelectedValues>
<SelectedValues>Misc</SelectedValues>
<SelectedValues>Postmaster</SelectedValues>
<SelectedValues>Raw</SelectedValues>
</UseAsXvalue>
<Valid>1</Valid>
</otrs_stats>

```

Now, that all subroutines are explained, this is the complete sample stats module.

```
# --
```

```

# Kernel/System/Stats/Dynamic/DynamicStatsTemplate.pm - all advice
  functions
# Copyright (C) 2001-2010 OTRS AG, http://otrs.org/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::Stats::Dynamic::DynamicStatsTemplate;

use strict;
use warnings;

use Kernel::System::Queue;
use Kernel::System::State;
use Kernel::System::Ticket;

use vars qw($VERSION);
$VERSION = qw($Revision: 1.1 $) [1];

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # check needed objects
    for my $Object (
        qw(DBObject ConfigObject LogObject UserObject TimeObject
MainObject EncodeObject)
    )
    {
        $Self->{$Object} = $Param{$Object} || die "Got no $Object!";
    }

    # created needed objects
    $Self->{QueueObject} = Kernel::System::Queue->new( %{$Self} );
    $Self->{TicketObject} = Kernel::System::Ticket->new( %{$Self} );
    $Self->{StateObject} = Kernel::System::State->new( %{$Self} );

    return $Self;
}

sub GetObjectName {
    my ( $Self, %Param ) = @_;

    return 'Sample Statistics';
}

sub GetObjectAttributes {
    my ( $Self, %Param ) = @_;

```

```

# get state list
my %StateList = $Self->{StateObject}->StateList(
    UserID => 1,
);

# get queue list
my %QueueList = $Self->{QueueObject}->GetAllQueues();

# get current time to fix bug#3830
my $TimeStamp = $Self->{TimeObject}->CurrentTimestamp();
my ($Date) = split /\s+/, $TimeStamp;
my $Today = sprintf "%s 23:59:59", $Date;

my @ObjectAttributes = (
    {
        Name           => 'State',
        UseAsXvalue    => 1,
        UseAsValueSeries => 1,
        UseAsRestriction => 1,
        Element        => 'StateIDs',
        Block           => 'MultiSelectField',
        Values          => \%StateList,
    },
    {
        Name           => 'Created in Queue',
        UseAsXvalue    => 1,
        UseAsValueSeries => 1,
        UseAsRestriction => 1,
        Element        => 'CreatedQueueIDs',
        Block           => 'MultiSelectField',
        Translation    => 0,
        Values          => \%QueueList,
    },
    {
        Name           => 'Create Time',
        UseAsXvalue    => 1,
        UseAsValueSeries => 1,
        UseAsRestriction => 1,
        Element        => 'CreateTime',
        TimePeriodFormat => 'DateInputFormat',      #
'DateInputFormatLong',
        Block           => 'Time',
        TimeStop        => $Today,
        Values          => {
            TimeStart => 'TicketCreateTimeNewerDate',
            TimeStop  => 'TicketCreateTimeOlderDate',
        },
    },
);

return @ObjectAttributes;
}

sub GetStatElement {

```

```

my ( $Self, %Param ) = @_;

# search tickets
return $Self->{TicketObject}->TicketSearch(
    UserID      => 1,
    Result      => 'COUNT',
    Permission  => 'ro',
    Limit       => 100_000_000,
    %Param,
);
}

sub ExportWrapper {
my ( $Self, %Param ) = @_;

# wrap ids to used spelling
for my $Use (qw(UseAsValueSeries UseAsRestriction UseAsXvalue)) {
    ELEMENT:
    for my $Element ( @{ $Param{$Use} } ) {
        next ELEMENT if !$Element || !$Element->{SelectedValues};
        my $ElementName = $Element->{Element};
        my $Values      = $Element->{SelectedValues};

        if ( $ElementName eq 'QueueIDs' || $ElementName eq
'CreatedQueueIDs' ) {
            ID:
            for my $ID ( @{$Values} ) {
                next ID if !$ID;
                $ID->{Content} = $Self->{QueueObject}-
>QueueLookup( QueueID => $ID->{Content} );
            }
        }
        elsif ( $ElementName eq 'StateIDs' || $ElementName eq
'CreatedStateIDs' ) {
            my %StateList = $Self->{StateObject}-
>StateList( UserID => 1 );
            ID:
            for my $ID ( @{$Values} ) {
                next ID if !$ID;
                $ID->{Content} = $StateList{ $ID->{Content} };
            }
        }
    }
}
return \%Param;
}

sub ImportWrapper {
my ( $Self, %Param ) = @_;

# wrap used spelling to ids
for my $Use (qw(UseAsValueSeries UseAsRestriction UseAsXvalue)) {
    ELEMENT:
    for my $Element ( @{ $Param{$Use} } ) {

```

```

        next ELEMENT if !$Element || !$Element->{SelectedValues};
        my $ElementName = $Element->{Element};
        my $Values      = $Element->{SelectedValues};

        if ( $ElementName eq 'QueueIDs' || $ElementName eq
'CreatedQueueIDs' ) {
            ID:
            for my $ID ( @{$Values} ) {
                next ID if !$ID;
                if ( $Self->{QueueObject}->QueueLookup( Queue =>
$ID->{Content} ) ) {
                    $ID->{Content}
                        = $Self->{QueueObject}->QueueLookup( Queue
=> $ID->{Content} );
                }
                else {
                    $Self->{LogObject}->Log(
                        Priority => 'error',
                        Message => "Import: Can' find the queue
$ID->{Content}!"
                    );
                    $ID = undef;
                }
            }
        }
        elsif ( $ElementName eq 'StateIDs' || $ElementName eq
'CreatedStateIDs' ) {
            ID:
            for my $ID ( @{$Values} ) {
                next ID if !$ID;

                my %State = $Self->{StateObject}->StateGet(
                    Name => $ID->{Content},
                    Cache => 1,
                );
                if ( $State{ID} ) {
                    $ID->{Content} = $State{ID};
                }
                else {
                    $Self->{LogObject}->Log(
                        Priority => 'error',
                        Message => "Import: Can' find state $ID-
>{Content}!"
                    );
                    $ID = undef;
                }
            }
        }
    }
    return \%Param;
}

1;

```

Configuration example

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<otrs_config version="1.0" init="Config">
  <ConfigItem
    Name="Stats::DynamicObjectRegistration###DynamicStatsTemplate"
    Required="0" Valid="1">
    <Description Lang="en">Here you can decide if the common stats
    module may generate stats about the number of default tickets a
    requester created.</Description>
    <Group>Framework</Group>
    <SubGroup>Core::Stats</SubGroup>
    <Setting>
      <Hash>
        <Item
          Key="Module">Kernel::System::Stats::Dynamic::DynamicStatsTemplate</
          Item>
        </Hash>
      </Setting>
    </ConfigItem>
  </otrs_config>
```

Use case examples

Use cases.

Caveats and Warnings

If you have a lot of cells in the result table and the GetStatElement is quite complex, the request can take a long time.

Release Availability

Dynamic stat modules are available since OTRS 2.0.

Static Stats

The subsequent paragraphs describe the static stats. Static stats are very easy to create as these modules have to implement only three subroutines.

- new
- Param
- Run

Code example

The following paragraphs describe the subroutines needed in a static stats.

```
sub new {
```

```

my ( $Type, %Param ) = @_;

# allocate new hash for object
my $Self = {%Param};
bless( $Self, $Type );

# check all needed objects
for my $Needed (
    qw(DBObject ConfigObject LogObject
       TimeObject MainObject EncodeObject)
)
{
    $Self->{$Needed} = $Param{$Needed} || die "Got no $Needed";
}

# create needed objects
$Self->{TypeObject} = Kernel::System::Type->new( %{$Self} );
$Self->{TicketObject} = Kernel::System::Ticket->new( %{$Self} );
$Self->{QueueObject} = Kernel::System::Queue->new( %{$Self} );

return $Self;
}

```

new creates a new instance of the static stats class. First it creates a new object and then it checks for the needed objects.

```

sub Param {
    my $Self = shift;

    my %Queues = $Self->{QueueObject}->GetAllQueues();
    my %Types = $Self->{TypeObject}->TypeList(
        Valid => 1,
    );

    my @Params = (
        {
            Frontend => 'Type',
            Name      => 'TypeIDs',
            Multiple  => 1,
            Size      => 3,
            Data      => \%Types,
        },
        {
            Frontend => 'Queue',
            Name      => 'QueueIDs',
            Multiple  => 1,
            Size      => 3,
            Data      => \%Queues,
        },
    );

    return @Params;
}

```

```

}
```

The Param method provides the list of all parameters/attributes that can be selected to create a static stat. It gets some parameters passed: The values for the stats attributes provided in a request, the format of the stats and the name of the object (name of the module).

The parameters/attributes have to be hashreferences with these key-value-pairs.

- Frontent
 - the label in the web interface
- Name
 - the HTML fieldname
- Data
 - the values shown in the attribute

Other parameter for the BuildSelection method of the LayoutObject can be used, as it is done with "Size" and "Multiple" in this sample module.

```

sub Run {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    for my $Needed (qw(TypeIDs QueueIDs)) {
        if ( !$Param{$Needed} ) {
            $Self->{LogObject}->Log(
                Priority => 'error',
                Message => "Need $Needed!",
            );
            return;
        }
    }

    # set report title
    my $Title = 'Tickets per Queue';

    # table headlines
    my @HeadData = (
        'Ticket Number',
        'Queue',
        'Type',
    );

    my @Data;
    my @TicketIDs = $Self->{TicketObject}->TicketSearch(
        UserID      => 1,
        Result       => 'ARRAY',
        Permission   => 'ro',
        %Param,
    );
}
```

```

    for my $TicketID ( @TicketIDs ) {
        my %Ticket = $Self->{TicketObject}->TicketGet (
            UserID => 1,
            TicketID => $TicketID,
        );
        push @Data, [ $Ticket{TicketNumber}, $Ticket{Queue},
$Ticket{Type} ];
    }

    return ( [$Title], [@HeadData], @Data );
}

```

The Run method actually generates the table data for the stats. It gets the attributes for this stats passed. In this sample it in %Param a key 'TypeIDs' and a key 'QueueIDs' exist (see attributes in Param method) and their values are arrayreferences. The returned data consists of three parts: Two arrayreferences and an array. In the first arrayreference the title for the statistic is stored, the second arrayreference contains the headlines for the columns in the table. And then the data for the table body follow.

```

# --
# Kernel/System/Stats/Static/StaticStatsTemplate.pm
# Copyright (C) 2001-2010 OTRS AG, http://otrs.org/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::Stats::Static::StaticStatsTemplate;

use strict;
use warnings;

use Kernel::System::Type;
use Kernel::System::Ticket;
use Kernel::System::Queue;

use vars qw($VERSION);
$VERSION = qw($Revision: 1.1 $) [1];

=head1 NAME

StaticStatsTemplate.pm - the module that creates the stats about
tickets in a queue

=head1 SYNOPSIS

All functions

=head1 PUBLIC INTERFACE

```

=over 4

=cut

=item new()

create an object

```

use Kernel::Config;
use Kernel::System::Encode;
use Kernel::System::Log;
use Kernel::System::Main;
use Kernel::System::Time;
use Kernel::System::DB;
use Kernel::System::Stats::Static::StaticStatsTemplate;

my $ConfigObject = Kernel::Config->new();
my $EncodeObject = Kernel::System::Encode->new(
    ConfigObject => $ConfigObject,
);
my $LogObject    = Kernel::System::Log->new(
    ConfigObject => $ConfigObject,
);
my $MainObject  = Kernel::System::Main->new(
    ConfigObject => $ConfigObject,
    LogObject    => $LogObject,
);
my $TimeObject  = Kernel::System::Time->new(
    ConfigObject => $ConfigObject,
    LogObject    => $LogObject,
);
my $DBObject    = Kernel::System::DB->new(
    ConfigObject => $ConfigObject,
    LogObject    => $LogObject,
    MainObject   => $MainObject,
);
my $StatsObject =
Kernel::System::Stats::Static::StaticStatsTemplate->new(
    ConfigObject => $ConfigObject,
    LogObject    => $LogObject,
    MainObject   => $MainObject,
    TimeObject   => $TimeObject,
    DBObject     => $DBObject,
    EncodeObject => $EncodeObject,
);

```

=cut

```

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {%Param};
    bless( $Self, $Type );
}

```

```

# check all needed objects
for my $Needed (
    qw(DBObject ConfigObject LogObject
        TimeObject MainObject EncodeObject)
    )
{
    $Self->{$Needed} = $Param{$Needed} || die "Got no $Needed";
}

# create needed objects
$Self->{TypeObject} = Kernel::System::Type->new( %{$Self} );
$Self->{TicketObject} = Kernel::System::Ticket->new( %{$Self} );
$Self->{QueueObject} = Kernel::System::Queue->new( %{$Self} );

return $Self;
}

=item Param()

Get all parameters a user can specify.

my @Params = $StatsObject->Param();

=cut

sub Param {
    my $Self = shift;

    my %Queues = $Self->{QueueObject}->GetAllQueues();
    my %Types = $Self->{TypeObject}->TypeList(
        Valid => 1,
    );

    my @Params = (
        {
            Frontend => 'Type',
            Name => 'TypeIDs',
            Multiple => 1,
            Size => 3,
            Data => \%Types,
        },
        {
            Frontend => 'Queue',
            Name => 'QueueIDs',
            Multiple => 1,
            Size => 3,
            Data => \%Queues,
        },
    );

    return @Params;
}

```

```
=item Run()

generate the statistic.

    my $StatsInfo = $StatsObject->Run(
        TypeIDs => [
            1, 2, 4
        ],
        QueueIDs => [
            3, 4, 6
        ],
    );

=cut

sub Run {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    for my $Needed (qw(TypeIDs QueueIDs)) {
        if ( !$Param{$Needed} ) {
            $Self->{LogObject}->Log(
                Priority => 'error',
                Message => "Need $Needed!",
            );
            return;
        }
    }

    # set report title
    my $Title = 'Tickets per Queue';

    # table headlines
    my @HeadData = (
        'Ticket Number',
        'Queue',
        'Type',
    );

    my @Data;
    my @TicketIDs = $Self->{TicketObject}->TicketSearch(
        UserID => 1,
        Result => 'ARRAY',
        Permission => 'ro',
        %Param,
    );

    for my $TicketID ( @TicketIDs ) {
        my %Ticket = $Self->{TicketObject}->TicketGet(
            UserID => 1,
            TicketID => $TicketID,
        );
        push @Data, [ $Ticket{TicketNumber}, $Ticket{Queue},
$Ticket{Type} ];
    }
}
```

```
    }  
    return ( [$Title], [@HeadData], @Data );  
  }  
1;  
  
=back  
  
=head1 TERMS AND CONDITIONS  
  
This software is part of the OTRS project (http://otrs.org/).  
  
This software comes with ABSOLUTELY NO WARRANTY. For details, see  
the enclosed file COPYING for license information (AGPL). If you  
did not receive this file, see http://www.gnu.org/licenses/agpl.txt.  
  
=head1 VERSION  
  
$Revision: 1.1 $ $Date: 2010/08/13 08:59:28 $  
  
=cut
```

Configuration example

There is no configuration needed. Right after installation, the module is available to create a statistic for this module.

Use case examples

Use cases.

Caveats and Warnings

Caveats and Warnings for static stats.

Release Availability

Static stat modules are available since OTRS 1.3.

Using old static stats

Standard OTRS versions 1.3 and 2.0 already facilitated the generation of stats. Various stats for OTRS versions 1.3 and 2.0 which have been specially developed to meet customers' requirements can be used in more recent versions too.

The files must merely be moved from the `Kernel/System/Stats/` path to `Kernel/System/Stats/Static/`. Additionally the package name of the respective script must be amended by `::Static`.

The following example shows how the first path is amended.

```
package Kernel::System::Stats::AccountedTime;
```

```
package Kernel::System::Stats::Static::AccountedTime;
```

Virtual Filesystem

The virtual filesystem is a layer to save files in a transparent way. This layer hides the logic how and where to save a file. In the subsequent paragraphs it is described how to write a new backend for the virtual filesystem. Currently two backends exist: DB and FS. The DB backend saves all files in the database and the FS backend saves the files in the "normal" filesystem.

The backend developed in this chapter uses a PDF file as a filesystem.

Code example

Configuration example

Use case examples

List of technical and subject-specific use cases.

Caveats and Warnings

A warning for the use of the DB backend. If you save all files in the database, the database can become quite big. This can impact database backups and recovery time.

Release Availability

List of known releases.

Ticket Number Generator Modules

Ticket number generators are used to create distinct identifiers aka TicketNumber for new tickets. Any method of creating a string of numbers is possible, you should use common sense about the length of the resulting string (guideline: 5-10). When creating a ticket number, make sure the result is prefixed by the SysConfig-Variable SystemID in order to enable the detection of ticket numbers on inbound email responses. A ticket number generator module needs the two functions TicketCreateNumber() and GetTNByString(). The method TicketCreateNumber() is called without parameters and returns the new ticket number. The method GetTNByString() is called with the param String which contains the string to be parsed for a ticket number and returns the ticket number if found.

Code example

See Kernel/System/Ticket/Number/UserRandom.pm in the package TemplateModule.

Configuration example

See Kernel/Config/Files/TicketNumberGenerator.xml in the package TemplateModule.

Use Cases

Ticket numbers should follow a specific scheme.

You will need to create a new ticket number generator if the default modules don't provide the ticket number scheme you'd like to use.

Caveats and Warnings

You should stick to the code of GetTNByString() as used in existing ticket number generators to prevent problems with ticket number parsing. Also the routine to detect a loop in TicketCreateNumber() should be kept intact to prevent duplicate ticket numbers.

Release Availability

Ticket number generators have been available in OTRS since OTRS 1.1.

Frontend Modules

Dashboard Module

Dashboard module to display statistics in the form of a line graph.

```
# --
# Kernel/Output/HTML/DashboardTicketStatsGeneric.pm - message of the
# day
# Copyright (C) 2001-2010 OTRS AG, http://otrs.org/
# --
# $Id: dashboard.xml,v 1.1 2010/08/13 08:59:28 mg Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::Output::HTML::DashboardTicketStatsGeneric;

use strict;
use warnings;

use vars qw($VERSION);
$VERSION = qw($Revision: 1.1 $) [1];

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {%Param};
    bless( $Self, $Type );
}
```

```

# get needed objects
for (
    qw(Config Name ConfigObject LogObject DBObject LayoutObject
ParamObject TicketObject UserID)
)
{
    die "Got no $_!" if !$Self->{$_};
}

return $Self;
}

sub Preferences {
    my ( $Self, %Param ) = @_;

    return;
}

sub Config {
    my ( $Self, %Param ) = @_;

    my $Key = $Self->{LayoutObject}->{UserLanguage} . '-' . $Self->{Name};
    return (
        %{ $Self->{Config} },
        CacheKey => 'TicketStats' . '-' . $Self->{UserID} . '-' .
$Key,
    );
}

sub Run {
    my ( $Self, %Param ) = @_;

    my %Axis = (
        '7Day' => {
            0 => { Day => 'Sun', Created => 0, Closed => 0, },
            1 => { Day => 'Mon', Created => 0, Closed => 0, },
            2 => { Day => 'Tue', Created => 0, Closed => 0, },
            3 => { Day => 'Wed', Created => 0, Closed => 0, },
            4 => { Day => 'Thu', Created => 0, Closed => 0, },
            5 => { Day => 'Fri', Created => 0, Closed => 0, },
            6 => { Day => 'Sat', Created => 0, Closed => 0, },
        },
    );

    my @Data;
    my $Max = 1;
    for my $Key ( 0 .. 6 ) {

        my $TimeNow = $Self->{TimeObject}->SystemTime();
        if ($Key) {
            $TimeNow = $TimeNow - ( 60 * 60 * 24 * $Key );

```

```

}
my ( $Sec, $Min, $Hour, $Day, $Month, $Year, $WeekDay )
  = $Self->{TimeObject}->SystemTime2Date(
    SystemTime => $TimeNow,
  );

>Data[$Key]->{Day} = $Self->{LayoutObject}->{LanguageObject}-
>Get (
  $Axis{'7Day'}->{$WeekDay}->{Day}
);

my $CountCreated = $Self->{TicketObject}->TicketSearch(

  # cache search result 20 min
  CacheTTL => 60 * 20,

  # tickets with create time after ... (ticket newer than
this date) (optional)
  TicketCreateTimeNewerDate => "$Year-$Month-$Day 00:00:00",

  # tickets with created time before ... (ticket older than
this date) (optional)
  TicketCreateTimeOlderDate => "$Year-$Month-$Day 23:59:59",

  CustomerID => $Param{Data}->{UserCustomerID},
  Result      => 'COUNT',

  # search with user permissions
  Permission => $Self->{Config}->{Permission} || 'ro',
  UserID    => $Self->{UserID},
);
>Data[$Key]->{Created} = $CountCreated;
if ( $CountCreated > $Max ) {
  $Max = $CountCreated;
}

my $CountClosed = $Self->{TicketObject}->TicketSearch(

  # cache search result 20 min
  CacheTTL => 60 * 20,

  # tickets with create time after ... (ticket newer than
this date) (optional)
  TicketCloseTimeNewerDate => "$Year-$Month-$Day 00:00:00",

  # tickets with created time before ... (ticket older than
this date) (optional)
  TicketCloseTimeOlderDate => "$Year-$Month-$Day 23:59:59",

  CustomerID => $Param{Data}->{UserCustomerID},
  Result      => 'COUNT',

  # search with user permissions
  Permission => $Self->{Config}->{Permission} || 'ro',

```

```

        UserID => $Self->{UserID},
    );
    $Data[$Key]->{Closed} = $CountClosed;
    if ( $CountClosed > $Max ) {
        $Max = $CountClosed;
    }
}

@Data = reverse @Data;
my $Source = $Self->{LayoutObject}->JSON(
    Data => \@Data,
);

my $Content = $Self->{LayoutObject}->Output(
    TemplateFile => 'AgentDashboardTicketStats',
    Data => {
        %{ $Self->{Config} },
        Key => int rand 99999,
        Max => $Max,
        Source => $Source,
    },
);

return $Content;
}

1;

```

To use this module add the following to the `Kernel/Config.pm` and restart your webserver (if you use `mod_perl`).

```

<ConfigItem Name="DashboardBackend###0250-TicketStats" Required="0"
Valid="1">
    <Description Lang="en">Parameters for the dashboard backend.
    "Group" are used to restricted access to the plugin (e. g. Group:
    admin;group1;group2;). "Default" means if the plugin is enabled per
    default or if the user needs to enable it manually. "CacheTTL" means
    the cache time in minutes for the plugin.</Description>
    <Description Lang="de">Parameter für das Dashboard Backend.
    "Group" ist verwendet um den Zugriff auf das Plugin einzuschränken
    (z. B. Group: admin;group1;group2;). "Default" bedeutet ob das
    Plugin per default aktiviert ist oder ob dies der Anwender manuell
    machen muss. "CacheTTL" ist die Cache-Zeit in Minuten nach der das
    Plugin erneut aufgerufen wird.</Description>
    <Group>Ticket</Group>
    <SubGroup>Frontend::Agent::Dashboard</SubGroup>
    <Setting>
        <Hash>
            <Item
Key="Module">Kernel::Output::HTML::DashboardTicketStatsGeneric</Item>
            <Item Key="Title">7 Day Stats</Item>
            <Item Key="Created">1</Item>

```

```
<Item Key="Closed">1</Item>
<Item Key="Permission">rw</Item>
<Item Key="Block">ContentSmall</Item>
<Item Key="Group"></Item>
<Item Key="Default">1</Item>
<Item Key="CacheTTL">45</Item>
</Hash>
</Setting>
</ConfigItem>
```

Caveats and Warnings

An excessive number of days or individual lines may lead to performance degradation.

Release Availability

from 2.4.0

Notification Module

Notification modules are used to display a notification below the main navigation. You can write and register your own notification module. There are currently 5 ticket menus in the OTRS framework.

- AgentOnline
- AgentTicketEscalation
- CharsetCheck
- CustomerOnline
- UIDCheck

Code Example

The notification modules are located under Kernel/Output/HTML/TicketNotification*.pm. There is an example of a notify module below. Save it under Kernel/Output/HTML/TicketNotificationCustom.pm. You just need 2 functions: new() and Run().

```
# --
# Kernel/Output/HTML/NotificationCustom.pm
# Copyright (C) 2001-2010 OTRS AG, http://otrs.org/
# --
# $Id: notify.xml,v 1.1 2010/08/13 08:59:28 mg Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::Output::HTML::NotificationCustom;

use strict;
```

```

use warnings;

use Kernel::System::Custom;

use vars qw($VERSION);
$VERSION = qw($Revision: 1.1 $) [1];

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # get needed objects
    for my $Object (qw(ConfigObject LogObject DBObject LayoutObject
TimeObject UserID)) {
        $Self->{$Object} = $Param{$Object} || die "Got no $Object!";
    }
    $Self->{CustomObject} = Kernel::System::Custom->new(%Param);
    return $Self;
}

sub Run {
    my ( $Self, %Param ) = @_;

    # get session info
    my %CustomParam = ();
    my @Customs = $Self->{CustomObject}->GetAllCustomIDs();
    my $IdleMinutes = $Param{Config}->{IdleMinutes} || 60 * 2;
    for (@Customs) {
        my %Data = $Self->{CustomObject}->GetCustomIDDData( CustomID =>
$_, );
        if (
            $Self->{UserID} ne $Data{UserID}
            && $Data{UserType} eq 'User'
            && $Data{UserLastRequest}
            && $Data{UserLastRequest} + ( $IdleMinutes * 60 ) > $Self-
>{TimeObject}->SystemTime()
            && $Data{UserFirstname}
            && $Data{UserLastname}
        )
        {
            $CustomParam{ $Data{UserID} } = "$Data{UserFirstname}
$Data{UserLastname}";
            if ( $Param{Config}->{ShowEmail} ) {
                $CustomParam{ $Data{UserID} } .=
" ($Data{UserEmail})";
            }
        }
    }
    for ( sort { $CustomParam{$a} cmp $CustomParam{$b} } keys
%CustomParam ) {
        if ( $Param{Message} ) {

```

```

        $Param{Message} .= ', ';
    }
    $Param{Message} .= "$CustomParam{$_}";
}
if ( $Param{Message} ) {
    return $Self->{LayoutObject}->Notify( Info => 'Custom Message:
%s", "' . $Param{Message} );
}
else {
    return '';
}
}
}
1;

```

Configuration Example

There is the need to activate your custom notification module. This can be done using the xml configuration below. There may be additional parameters in the config hash for your notification module.

```

<ConfigItem Name="Frontend::NotifyModule###3-Custom" Required="0"
Valid="0">
    <Description Lang="en">Module to show custom message in the agent
interface.</Description>
    <Description Lang="de">Mit diesem Modul können eigene Meldungenen
innerhalb des Agent-Interfaces angezeigt werden.</Description>
    <Group>Framework</Group>
    <SubGroup>Frontend::Agent::ModuleNotify</SubGroup>
    <Setting>
        <Hash>
            <Item
Key="Module">Kernel::Output::HTML::NotificationCustom</Item>
            <Item Key="Key1">1</Item>
            <Item Key="Key2">2</Item>
        </Hash>
    </Setting>
</ConfigItem>

```

Use Case Example

Useful ticket menu implementation could be a link to a external tool if parameters (e.g. FreeTextField) have been set.

Release Availability

Name	Release
NotificationAgentOnline	2.0
NotificationAgentTicketEscalation	2.0
NotificationCharsetCheck	1.2

Name	Release
NotificationCustomerOnline	2.0
NotificationUIDCheck	1.2

Ticket Menu Module

Ticket menu modules are used to display an additional link in the menu above a ticket. You can write and register your own ticket menu module. There are 4 ticket menus (Generic, Lock, Responsible and TicketWatcher) which come with the OTRS framework. For more information please have a look at the OTRS admin manual.

Code Example

The ticket menu modules are located under Kernel/Output/HTML/TicketMenu*.pm. There is an example of a ticket-menu module below. Save it under Kernel/Output/HTML/TicketMenuCustom.pm. You just need 2 functions: new() and Run().

```
# --
# Kernel/Output/HTML/TicketMenuCustom.pm
# Copyright (C) 2001-2010 OTRS AG, http://otrs.org/
# --
# Id: TicketMenuCustom.pm,v 1.17 2010/04/12 21:34:06 martin Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::Output::HTML::TicketMenuCustom;

use strict;
use warnings;

use vars qw($VERSION);
$VERSION = qw($Revision: 1.1 $) [1];

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # get needed objects
    for my $Object (qw(ConfigObject LogObject DBObject LayoutObject
        UserID TicketObject)) {
        $Self->{$Object} = $Param{$Object} || die "Got no $Object!";
    }

    return $Self;
}
}
```

```

sub Run {
    my ( $Self, %Param ) = @_;

    # check needed stuff
    if ( !$Param{Ticket} ) {
        $Self->{LogObject}->Log(
            Priority => 'error',
            Message => 'Need Ticket!'
        );
        return;
    }

    # check if frontend module registered, if not, do not show action
    if ( $Param{Config}->{Action} ) {
        my $Module = $Self->{ConfigObject}->Get('Frontend::Module')-
>{ $Param{Config}->{Action} };
        return if !$Module;
    }

    # check permission
    my $AccessOk = $Self->{TicketObject}->Permission(
        Type => 'rw',
        TicketID => $Param{Ticket}->{TicketID},
        UserID => $Self->{UserID},
        LogNo => 1,
    );
    return if !$AccessOk;

    # check permission
    if ( $Self->{TicketObject}->CustomIsTicketCustom( TicketID =>
$Param{Ticket}->{TicketID} ) ) {
        my $AccessOk = $Self->{TicketObject}->OwnerCheck(
            TicketID => $Param{Ticket}->{TicketID},
            OwnerID => $Self->{UserID},
        );
        return if !$AccessOk;
    }

    # check acl
    return
        if defined $Param{ACL}->{ $Param{Config}->{Action} }
        && !$Param{ACL}->{ $Param{Config}->{Action} };

    # if ticket is customized
    if ( $Param{Ticket}->{Custom} eq 'lock' ) {

        # if it is locked for somebody else
        return if $Param{Ticket}->{OwnerID} ne $Self->{UserID};

        # show custom action
        return {
            %{ $Param{Config} },
            %{ $Param{Ticket} },
            %Param,
        }
    }
}

```

```

        Name          => 'Custom',
        Description => 'Custom to give it back to the queue!',
        Link          =>
'Action=AgentTicketCustom;Subaction=Custom;TicketID=
$QData{"TicketID"}',
    };
}

# if ticket is customized
return {
    %{ $Param{Config} },
    %{ $Param{Ticket} },
    %Param,
    Name          => 'Custom',
    Description => 'Custom it to work on it!',
    Link          =>
'Action=AgentTicketCustom;Subaction=Custom;TicketID=
$QData{"TicketID"}',
    };
}
1;

```

Configuration Example

There is the need to activate your custom ticket menu module. This can be done using the xml configuration below. There may be additional parameters in the config hash for your ticket menu module.

```

<ConfigItem Name="Ticket::Frontend::MenuModule###110-Custom"
  Required="0" Valid="1">
  <Description Lang="en">Module to show custom link in menu.</
Description>
  <Description Lang="de">Mit diesem Modul wird der Custom-Link in
der Linkleiste der Ticketansicht angezeigt.</Description>
  <Group>Ticket</Group>
  <SubGroup>Frontend::Agent::Ticket::MenuModule</SubGroup>
  <Setting>
  <Hash>
  <Item
Key="Module">Kernel::Output::HTML::TicketMenuCustom</Item>
  <Item Key="Name">Custom</Item>
  <Item Key="Action">AgentTicketCustom</Item>
  </Hash>
  </Setting>
</ConfigItem>

```

Use Case Example

Useful ticket menu implementation could be a link to a external tool if parameters (e.g. FreeTextField) have been set.

Caveats and Warnings

The ticket menu directs to an URL that can be handled. If you want to handle that request via the OTRS framework, you have to write your own frontend module.

Release Availability

Name	Release
TicketMenuGeneric	2.0
TicketMenuLock	2.0
TicketMenuResponsible	2.1
TicketMenuTicketWatcher	2.4

Old Module Descriptions

Please remove these old sections if newer ones were created.

Navigation Module

In this module layer you can create dynamic navigation bar items with dynamic content (Name and Description). Navigation Module are located under Kernel/Output/HTML/NavBar*.pm.

Format:

```
# --
# Kernel/Output/HTML/NavBarABC.pm - shows a navbar item dynamically
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: module-format.xml,v 1.2 2010/12/01 13:22:15 mg Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::Output::HTML::NavBarABC;

use strict;
use warnings;

# --
sub new {
    my ( $Type, %Param ) = @_;
    [...]
    return $Self;
}
# --
sub Run {
    my ( $Self, %Param ) = @_;
    my %Return = ();
    $Return{'0999989'} = {
```

```

        Block      => 'ItemPersonal',
        Description => 'Some Descripton',
        Name       => 'Text',
        Image      => 'new-message.png',
        Link       => 'Action=AgentMailbox&Subaction=New',
        AccessKey  => 'j',
    };
    return %Return;
}
# --
1;

```

To use this module add the following code to the Kernel/Config.pm and restart your webserver (if you use mod_perl).

```

[Kernel/Config.pm]
# agent interface notification module
$Self->{'Frontend::NavBarModule'}->{'99-ABC'} = {
    Module => 'Kernel::Output::HTML::NavBarABC',
};

```

Frontend Modules

Frontend Modules are located under "\$OTRS_HOME/Kernel/Modules/*.pm". There are two public functions in there - new() and run() - which are accessed from the Frontend Handle (e. g. index.pl). "new()" is used to create a frontend module object. The Frontend Handle provides the used frontend module with the basic framework object. These are, for example: ParamObject (to get formular params), DBObject (to use existing database connects), LayoutObject (to use templates and other html layout functions), ConfigObject (to access config settings), LogObject (to use the framework log system), UserObject (to get the user functions from the current user), GroupObject (to get the group functions), MainObject (to get main functions like 'Require') and EncodeObject (for the charset encoding).

For more information on core modules see <http://dev.otrs.org/>

Format:

```

# --
# Kernel/Modules/AgentTest.pm - message of the day
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: module-format.xml,v 1.2 2010/12/01 13:22:15 mg Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::Modules::AgentTest;

```

```

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.2 $) [1];

# --
sub new {
    my ( $Type, %Param ) = @_;
    [...]
    return $Self;
}
# --
sub Run {
    my ( $Self, %Param ) = @_;
    [...]
    # ----- #
    # add a new object (Note: dtl text 'New')
    # ----- #
    if ($Self->{Subaction} eq 'Add') {
        my $Output = '';
        my %Frontend = ();
        [...]
        # add add block
        $Self->{LayoutObject}->Block(
            Name => 'Add',
            Data => {%Param, %Frontend},
        );
        # build output
        $Output .= $Self->{LayoutObject}->Header(Area => 'Agent',
Title => "Test");
        $Output .= $Self->{LayoutObject}->NavigationBar();
        $Output .= $Self->{LayoutObject}->Output(
            Data => {%Param, %Frontend},
            TemplateFile => 'AgentTest',
        );
        $Output .= $Self->{LayoutObject}->Footer();
        return $Output;
    }
    # ----- #
    # show overview screen
    # ----- #
    elsif ($Self->{Subaction} eq 'Overview') {
        # add overview block
        $Self->{LayoutObject}->Block(
            Name => 'Overview',
            Data => {%Param, %Frontend},
        );
        # build output
        $Output .= $Self->{LayoutObject}->Header(Area => 'Agent',
Title => "Test");
        $Output .= $Self->{LayoutObject}->NavigationBar();
        $Output .= $Self->{LayoutObject}->Output(
            Data => {%Param, %Frontend},

```

```

        TemplateFile => 'AgentTest',
    );
    $Output .= $Self->{LayoutObject}->Footer();
    return $Output;
}
# ----- #
# show error screen
# ----- #
return $Self->{LayoutObject}->ErrorScreen(Message => "Invalid
Subaction process!");
}
# --
1;

```

You also need a module registration for frontend modules. Define read only groups with the "GroupRo" and read/write groups with the 'Group' param (see table below for details). You can define navigation bar icons via the "NavBar"param, too (see table below for details).

```

[Kernel/Config.pm]
$Self->{'Frontend::Module'}->{'AgentTest'} = {
    Group => ['admin'],
    GroupRo => ['test', 'admin'],
    Description => 'A test Module',
    NavBarName => 'Ticket',
    NavBar => [
        {
            Description => 'Test Module',
            Name => 'Test',
            Image => 'stats.png',
            Link => 'Action=AgentTest',
            NavBar => 'Ticket',
            Prio => 85,
        },
    ],
};

```

You can access this frontend module via http (browse) with the Action param = Module or over the navigation bar.

<http://localhost/otrs/index.pl?Action=AgentTest> []

Description of Frontend::Module options:

Key	Description
Group	An ARRAY reference of rw groups of this module.
GroupRo	An ARRAY reference of ro groups of this module.
Description	Module description, just for internal use - not shown in the user interface.

Key	Description
NavBarName	NavBar context name of this module.

Description of NavBar (icon points) options:

Key	Description
Description	The description of the icon which is shown in the navbar after the curser is pointed on it.
Name	The icon name shown in the navbar.
Image	The icon image shown in the navbar.
Link	The link behind the icon in the navbar.
NavBar	Only shown this icon in this NavBar context.
Prio	Sort prio of the icon in the navbar.

Core Modules

Core modules are located under \$OTRS_HOME/Kernel/System/*. This layer is for the logical work. The modules are used to handle system routines like "lock ticket" and "create ticket". These modules always need pod (Perl Documentation).

A few common core modules are: Log (Kernel::System::Log); Ticket (Kernel::System::Ticket), Auth (Kernel::System::Auth), User (Kernel::System::User), Email (Kernel::System::Email).

For more information on the core modules see <http://dev.otrs.org> [<http://dev.otrs.org/>]

Format:

```
# --
# Kernel/System/Backend.pm - a core module
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: module-format.xml,v 1.2 2010/12/01 13:22:15 mg Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::Backend;

use strict;
use warnings;

=head1 NAME

Kernel::System::Log - global log interface

=head1 SYNOPSIS

All log functions.
```

```
=head1 PUBLIC INTERFACE

=over 4

=item new()

create a backend object

use Kernel::Config;
use Kernel::System::Backend;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.2 $) [1];

my $ConfigObject = Kernel::Config->new();
my $BackendObject = Kernel::System::Backend->new(ConfigObject =>
    $ConfigObject);

=cut

sub new {
    my ( $Type, %Param ) = @_;
    [...]
    return $Self;
}

=item SomeMethodA()

some info about the methode

$BackendObject->SomeMethodA(ParamA => 'error', ParamB => "Need
    something!");

=cut

sub SomeMethodA{
    my ( $Self, %Param ) = @_;
    [...]
    return 1;
}
1;

=head1 TERMS AND CONDITIONS

This software is part of the OTRS project (http://otrs.org/).

This software comes with ABSOLUTELY NO WARRANTY. For details, see
the enclosed file COPYING for license information (AGPL). If you
did not receive this file, see http://www.gnu.org/licenses/agpl.txt.

=head1 VERSION

$Revision: 1.2 $ $Date: 2010/12/01 13:22:15 $
```

```
=cut
```

Customer User Module

This module layer can be used as a bridge between your customer source system and OTRS. Thus it is possible to use your customer data directly for your data ware house (read only and read/write).

Format:

```
# --
# Kernel/System/CustomerUser/ABC.pm - a customer data backend
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: module-format.xml,v 1.2 2010/12/01 13:22:15 mg Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::CustomerUser::ABD;

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.2 $) [1];

# --
sub new {
    my ( $Type, %Param ) = @_;
    [...]
    return $Self;
}
# --
sub CustomerName {
    [...]
    return $Name;
}
# --
sub CustomerSearch {
    [...]
    return %Result;
}
# --
sub CustomerUserList {
    [...]
    return %List;
}
# --
sub CustomerIDs {
    [...]
```

```

        return @CustomerIDs;
    }
    # --
    sub CustomerUserDataGet {
        [...]
        return %Data;
    }
    # --
    sub CustomerUserAdd {
        [...]
        return 1
    }
    # --
    sub CustomerUserUpdate {
        [...]
        return 1;
    }
    # --
    sub SetPassword {
        [...]
        return 1;
    }
    }
    1;

```

To use this module, see the admin manual.

Customer Navigation Module

In this module layer you can create dynamic navigation bar items with dynamic content (Name and Description).

The format is the same as in the Navigation Module.

Just the config setting key is different. To use this module, add the following to the Kernel/Config.pm and restart your webserver (if you use mod_perl).

```

[Kernel/Config.pm]
# customer notification module
$self->{'CustomerFrontend::NavBarModule'}->{'99-ABC'} = {
    Module => 'Kernel::Output::HTML::NavBarABC',
};

```

Ticket Modules

Ticket Number Module

If you want to create your own ticket number format, just create your own ticket number module. These modules are located under "Kernel/System/Ticket/Number/*.pm". For default modules see the admin manual. You just need 2 functions: CreateTicketNr() and GetTNByString():

Format:

```

# --

```

```

# Ticket/Number/Simple.pm - a ticket number auto increment generator
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: module-format.xml,v 1.2 2010/12/01 13:22:15 mg Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::Ticket::Number::Simple;

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.2 $) [1];

sub CreateTicketNr {
    my $Self = shift;
    my $JumpCounter = shift || 0;
    # get needed config options
    [...]
    return $Tn;
}
# --
sub GetTNByString {
    my $Self = shift;
    my $String = shift || return;
    [...]
    return $Tn;
}
1;

```

Ticket PostMaster Module

PostMaster modules are used during the PostMaster process. There are two kinds of PostMaster modules. PostMasterPre (used after parsing an email) and PostMasterPost (used after an email is processed and in the database) modules.

If you want to create your own postmaster filter, just create your own module. These modules are located under "Kernel/System/PostMaster/Filter/*.pm". For default modules see the admin manual. You just need two functions: new() and Run():

The following is an exemplary module to match emails and set X-OTRS-Headers (see doc/X-OTRS-Headers.txt for more info).

Kernel/Config.pm:

```

# Job Name: 1-Match
# (block/ignore all spam email with From: noreply@)
$Self->{'PostMaster::PreFilterModule'}->{'1-Example'} = {
    Module => 'Kernel::System::PostMaster::Filter::Example',

```

```

Match => {
    From => 'noreply@',
},
Set => {
    'X-OTRS-Ignore' => 'yes',
},
};

```

Format:

```

# --
# Kernel/System/PostMaster/Filter/Example.pm - a postmaster filter
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: module-format.xml,v 1.2 2010/12/01 13:22:15 mg Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::PostMaster::Filter::Example;

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.2 $) [1];

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless ($Self, $Type);

    $Self->{Debug} = $Param{Debug} || 0;

    # get needed objects
    for (qw(ConfigObject EncodeObject LogObject DBObject)) {
        $Self->{$_} = $Param{$_} || die "Got no $_!";
    }

    return $Self;
}

sub Run {
    my ( $Self, %Param ) = @_;
    # get config options
    my %Config = ();
    my %Match = ();
    my %Set = ();

```

```

if ($Param{JobConfig} &&& ref($Param{JobConfig}) eq 'HASH')
{
    %Config = %{$Param{JobConfig}};
    if ($Config{Match}) {
        %Match = %{$Config{Match}};
    }
    if ($Config{Set}) {
        %Set = %{$Config{Set}};
    }
}
# match 'Match => ???' stuff
my $Matched = '';
my $MatchedNot = 0;
for (sort keys %Match) {
    if ($Param{GetParam}->{$_} &&& $Param{GetParam}->{$_}
=~ /$Match{$_}/i) {
        $Matched = $1 || '1';
        if ($Self->{Debug} > 1) {
            $Self->{LogObject}->Log(
                Priority => 'debug',
                Message => "'$Param{GetParam}->{$_}' =~ /
$Match{$_}/i matched!",
            );
        }
    }
    else {
        $MatchedNot = 1;
        if ($Self->{Debug} > 1) {
            $Self->{LogObject}->Log(
                Priority => 'debug',
                Message => "'$Param{GetParam}->{$_}' =~ /
$Match{$_}/i matched NOT!",
            );
        }
    }
}
# should I ignore the incoming mail?
if ($Matched &&& !$MatchedNot) {
    for (keys %Set) {
        if ($Set{$_} =~ /\[.*\.*\.*\]/i) {
            $Set{$_} = $Matched;
        }
        $Param{GetParam}->{$_} = $Set{$_};
        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message => "Set param '$_' to '$Set{$_}' (Message-ID:
$Param{GetParam}->{'Message-ID'}) ",
        );
    }
}
return 1;
}

```

1;

The following image shows you the email processing flow.

Ticket Menu Module

To add links in the ticket menu, just use ticket menu modules.

If you want to create your own ticket menu link, just create your own module. These modules are located under "Kernel/Output/HTML/TicketMenu*.pm". For default modules see the admin manual. You just need two functions: new() and Run():

The following example shows you how to show a lock or a unlock ticket link.

Kernel/Config.pm:

```
# for ticket zoom menu
$self->{'Ticket::Frontend::MenuModule'}->{'100-Lock'} = {
    Action => 'AgentTicketLock',
    Module => 'Kernel::Output::HTML::TicketMenuLock',
    Name   => 'Lock'
};

# for ticket preview menu
$self->{'Ticket::Frontend::PreMenuModule'}->{'100-Lock'} = {
    Action => 'AgentTicketLock',
    Module => 'Kernel::Output::HTML::TicketMenuLock',
    Name   => 'Lock'
};
```

Format:

```
# --
# Kernel/Output/HTML/TicketMenuLock.pm
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: module-format.xml,v 1.2 2010/12/01 13:22:15 mg Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::Output::HTML::TicketMenuLock;

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.2 $) [1];
```

```

# --
sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless ($Self, $Type);

    # get needed objects
    for (qw(ConfigObject LogObject DBObject LayoutObject UserID
TicketObject)) {
        $Self->{$_} = $Param{$_} || die "Got no $_!";
    }

    return $Self;
}
# --
sub Run {
    my ( $Self, %Param ) = @_;
    # check needed stuff
    if (!$Param{Ticket}) {
        $Self->{LogObject}->Log(Priority => 'error', Message => 'Need
Ticket!');
        return;
    }

    # check permission
    if ($Self->{TicketObject}->LockIsTicketLocked(TicketID =>
$Param{TicketID})) {
        my $AccessOk = $Self->{TicketObject}->OwnerCheck(
            TicketID => $Param{TicketID},
            OwnerID => $Self->{UserID},
        );
        if (!$AccessOk) {
            return $Param{Counter};
        }
    }

    $Self->{LayoutObject}->Block(
        Name => 'Menu',
        Data => { },
    );
    if ($Param{Counter}) {
        $Self->{LayoutObject}->Block(
            Name => 'MenuItemSplit',
            Data => { },
        );
    }
    if ($Param{Ticket}->{Lock} eq 'lock') {
        $Self->{LayoutObject}->Block(
            Name => 'MenuItem',
            Data => {
                %{$Param{Config}},
                %{$Param{Ticket}},
            },
        );
    }
}

```

```

        %Param,
        Name => 'Unlock',
        Description => 'Unlock to give it back to the queue!',
        Link =>
        'Action=AgentTicketLock&amp;Subaction=Unlock&amp;TicketID=
        $QData{"TicketID"}',
        },
    );
}
else {
    $Self->{LayoutObject}->Block(
        Name => 'MenuItem',
        Data => {
            %{$Param{Config}},
            %Param,
            Name => 'Lock',
            Description => 'Lock it to work on it!',
            Link =>
            'Action=AgentTicketLock&amp;Subaction=Lock&amp;TicketID=
            $QData{"TicketID"}',
            },
        );
}
$Param{Counter}++;
return $Param{Counter};
}
# --
1;

```

Ticket Event Module

Ticket event modules are running right after a ticket action takes place. Per convention these modules are located in the directory "Kernel/System/Ticket/Event". An ticket event module needs only the two functions new() and Run(). The method Run() receives at least the parameters Event, UserID, and TicketID. For the events TicketFreeTextUpdate and TicketFreeTimeUpdate, the parameter Counter is also passed to the Run() method, in order to identify which free field was updated. The article related handler functions also receives the ArticleID as parameter.

Code example

See Kernel/System/Ticket/Event/EventModulePostTemplate.pm in the package TemplateModule.

Configuration example

See Kernel/Config/Files/EventModulePostTemplate.xml in the package TemplateModule.

Use Cases

A ticket should be unlocked after a move action.

This standard feature has been implemented with the ticket event module Kernel::System::Ticket::Event::ForceUnlock. When this feature is not wanted, then it can be turned off by unsetting the SysConfig entry Ticket::EventModulePost###910-ForceUnlockOnMove.

Perform extra cleanup action when a ticket is deleted.

A customized OTRS might hold non-standard data in additional database tables. When a ticket is deleted then this additional data needs to be deleted. This functionality can be achieved with a ticket event module listening to 'TicketDelete' events.

New tickets should be twittered.

A ticket event module listening to 'TicketCreate' can send out tweets.

Caveats and Warnings

No caveats are known.

Release Availability

Ticket events have been available in OTRS since OTRS 2.0.

Ticket Events for OTRS 2.0: TicketCreate, TicketDelete, TicketTitleUpdate, TicketUnlockTimeoutUpdate, TicketEscalationStartUpdate, MoveTicket, SetCustomerData, TicketFreeTextSet, TicketFreeTimeSet, TicketPendingTimeSet, LockSet, StateSet, OwnerSet, TicketResponsibleUpdate, PrioritySet, HistoryAdd, HistoryDelete, TicketAccountTime, TicketMerge, ArticleCreate, ArticleFreeTextSet, ArticleUpdate, ArticleSend, ArticleBounce, SendAgentNotification, SendCustomerNotification, SendAutoResponse, ArticleFlagSet;

Ticket Events for OTRS 2.1 and higher: TicketCreate, TicketDelete, TicketTitleUpdate, TicketUnlockTimeoutUpdate, TicketEscalationStartUpdate, TicketQueueUpdate (MoveTicket), TicketCustomerUpdate (SetCustomerData), TicketFreeTextUpdate (TicketFreeTextSet), TicketFreeTimeUpdate (TicketFreeTimeSet), TicketPendingTimeUpdate (TicketPendingTimeSet), TicketLockUpdate (LockSet), TicketStateUpdate (StateSet), TicketOwnerUpdate (OwnerSet), TicketResponsibleUpdate, TicketPriorityUpdate (PrioritySet), TicketSubscribe, TicketUnsubscribe, HistoryAdd, HistoryDelete, TicketAccountTime, TicketMerge, ArticleCreate, ArticleFreeTextUpdate (ArticleFreeTextSet), ArticleUpdate, ArticleSend, ArticleBounce, ArticleAgentNotification (SendAgentNotification), ArticleCustomerNotification (SendCustomerNotification), ArticleAutoResponse (SendAutoResponse), ArticleFlagSet, ArticleFlagDelete;

Ticket Events for OTRS 2.4: TicketCreate, TicketDelete, TicketTitleUpdate, TicketUnlockTimeoutUpdate, TicketQueueUpdate(MoveTicket), TicketTypeUpdate, TicketServiceUpdate, TicketSLAUpdate, TicketCustomerUpdate (SetCustomerData), TicketFreeTextUpdate, TicketFreeTimeUpdate, TicketPendingTimeUpdate (TicketPendingTimeSet), TicketLockUpdate (LockSet), TicketStateUpdate (StateSet), TicketOwnerUpdate (OwnerSet), TicketResponsibleUpdate, TicketPriorityUpdate (PrioritySet), HistoryAdd, HistoryDelete, TicketAccountTime, TicketMerge, ArticleCreate, ArticleFreeTextUpdate (ArticleFreeTextSet), ArticleUpdate, ArticleSend, ArticleBounce, ArticleAgentNotification (SendAgentNotification), ArticleCustomerNotification (SendCustomerNotification), ArticleAutoResponse(SendAutoResponse), ArticleFlagSet, ArticleFlagDelete;

More Modules

The Agent Ticket Permission Modules (Kernel/System/Ticket/Permission/) contain functions to verify an agent's authorisation to access a ticket.

The Customer Ticket Permission Modules (Kernel/System/Ticket/CustomerPermission/) contain functions to verify a customer's authorisation to access a ticket.

The Article Module (Kernel/System/Ticket/Article.pm) facilitates the readout and generating of ticket articles.

More modules and their descriptions are listed under <http://dev.otrs.org/>

Object Basics

This chapter describes the basics of a new object (e. g. a ticket, faq, calendar, ...) and how the environment should look like.

Object Options

An object (e.g. a ticket, faq, calendar, ...) should at least have the following options (named after their function) in the application and in the database.

Application	database naming
ObjectID	object_id
Number	number
Title	title
...	...
StateID	state_id
WordAndWord	word_and_word
...	...
Created	created
CreatedBy	created_by
Changed	changed
ChangedBy	changed_by

Search Options

A search over free text fields should support:

- a normal search "thomas" (always)
- an and condition like "thomas+raith" (if possible)
- an or condition like "thomas||raith" (if possible)

Config Naming

Config naming should be with a leading prefix, the object name like this:

```
$Self->{"Object::Option"} = 1234;
```

Config-Hashes should be named with the same name as in the .dtl. For example:

```
$Self->{"Object::CategoryList"} -> $Data{"CategoryList"}
```

The config order should be global setting followed by detail settings.

Config File

An object should have a unique config file which should be located under \$OTRS_HOME/Kernel/Config/Files/*.pm. For example:

```
# module reg and nav bar
$self->{'Frontend::Module'}->{'AgentFileManager'} = {
    Description => 'Web File Manager',
    NavBarName => 'FileManager',
    NavBar => [
        {
            Description => 'A web file manager',
            Name => 'File-Manager',
            Image => 'filemanager.png',
            Link => 'Action=AgentFileManager',
            NavBar => 'FileManager',
            Prio => 5000,
            AccessKey => 'f',
        },
    ],
};

# browse/download root directory
$self->{"FileManager::Root"} = '/home/';

# trash directory
$self->{"FileManager::Trash"} = "/home/Trash/";
```

Description of the Config Preferences:

Element	description
NavBarName	module name
Group/GroupRo	group access authorization
Name	name of the link button
Image	image for the link button
Link	URI
NavBar	module name (correlation)
Prio	prio in the button list
AccessKey	short key (key + ALT) for quick access over the keyboard.

NavBar Settings

A NavBar item should look like the following example:

NavBarPoint	Prio	AccessKey	Image
Overview	100	o	overview.png

NavBarPoint	Prio	AccessKey	Image
New	200	n	new.png
Search	300	s	search.png
Delete	400	d	delete.png
Import	500	-	import.png
Setting	900	-	module_setting.png

Menu functions - generic, used by any application module

NavBarPoint	Prio	AccessKey	Image
Logout	10	l	logout.png
Preferences	0	p	preferences.png
New Messages	999989	m	new-messages.png
Locked Tickets	9999999	k	personal.png

Menu functions - global, always used

NavBarPoint	Prio	AccessKey	Image
Ticket	200	t	ticket.png
Incident (SIRIOS-Project)	2000	i	incident.png
Advisory (SIRIOS-Project)	2100	d	advisory.png
ShortAdvisory (SIRIOS-BSI-specific)	2150	z	advisory_short.png
VirusWarning (SIRIOS-BSI-specific)	2300	x	viruswarning.png
FreeTextMessage (SIRIOS-BSI-specific)	2400	y	freetextmessage.png
Vulnerability (SIRIOS-Project)	2500	v	vulnerability.png
Artefact (SIRIOS-Project)	2600	r	artefactdb.png
WebWatcher (SIRIOS-Project)	2700	z	webwatcher.png
IDMEFConsole (SIRIOS-Project)	2800	-	idmef_console.png
WID-Authoring (WID-Project)	2900	-	wid_authoring.png
WID-Portal-Admin-User (WID-Project)	2910	-	wid_portal_admin_user.png
WID-Portal-Admin-Group (WID-Project)	2920	-	wid_portal_admin_group.png
ITSMService (OTRS::ITSM)	3100	-	itsm_service.png

NavBarPoint	Prio	AccessKey	Image
ITSMConfigItem (OTRS::ITSM)	3200	-	itsm_configitem.png
ITSMLocation (OTRS::ITSM)	3300	-	itsm_location.png
TimeAccounting	6000	-	time_accounting.png
ContentManager	7050	-	contentmanager.png
Calendar	8000	c	calendar.png
FileManager	8100	f	filemanager.png
WebMail	8200	w	webmail.png
FAQ	8300	q	help.png
Call	8400	-	call.png
Stats	8500	-	stats.png
CustomerDB	9000	-	folder_yellow.png
CustomerCompanyDB	9100	-	folder_yellow.png
Admin	10000	a	admin.png

Table 3: Menu Applications -default application modules

Note

AccessKey "g" is also reserved for the submission of forms.

Screen flow

An object module should have the following process flow:

Writing an OTRS module for a new object

In this chapter, the writing of a new OTRS module is illustrated on the basis of a simple small programme. Necessary prerequisite is an OTRS development environment as specified in the chapter of the same name.

What we want to write

We want to write a little OTRS module that displays the text 'Hello World' when called up. First of all we must build the directory /Hello World for the module in the developer directory. In this directory, all directories existent in OTRS can be created. Each module should at least contain the following directories:

Kernel/

Kernel/System/

Kernel/Modules/

Kernel/Output/HTML/Standard/

Kernel/Config/

Kernel/Config/Files/

Kernel/Language/

Default Config File

The creation of a module registration facilitates the display of the new module in OTRS. Therefore we create a file '/Kernel/System/Config/Files/HelloWorld.xml'. In this file, we create a new config element. The impact of the various settings is described in the chapter 'Config Mechanism'.

```
<?xml version="1.0" encoding="UTF-8" ?>
<otrs_config version="1.0" init="Application">
  <ConfigItem Name="Frontend::Module###AgentHelloWorld"
    Required="1" Valid="1">
    <Description Lang="en">FrontendModuleRegistration for
    HelloWorld modul.</Description>
    <Description Lang="de">FrontendModulRegistration für das
    HelloWorld Modul.</Description>
    <Group>HelloWorld</Group>
    <SubGroup>AgentFrontendModuleRegistration</SubGroup>
    <Setting>
      <FrontendModuleReg>
        <Title>HelloWorld</Title>
        <Group>users</Group>
        <Description>HelloWorld</Description>
        <NavBarName>HelloWorld</NavBarName>
        <NavBar>
          <Description>HelloWorld</Description>
          <Name>HelloWorld</Name>
          <Image>overview.png</Image>
          <Link>Action=AgentHelloWorld</Link>
          <NavBar>HelloWorld</NavBar>
          <Type>Menu</Type>
          <Prio>8400</Prio>
          <Block>ItemArea</Block>
        </NavBar>
      </FrontendModuleReg>
    </Setting>
  </ConfigItem>
</otrs_config>
```

Frontend Module

After creating the links and executing the Sysconfig, a new module with the name 'HelloWorld' is displayed. When calling it up, an error message is displayed as OTRS cannot find the matching frontend module yet. This is the next thing to be created. To do so, we create the following file:

```
# --
# Kernel/Modules/AgentHelloWorld.pm - frontend modul
# Copyright (C) (year) (name of author) (email of author)
# --
```

```

# $Id: writing-otrs-application.xml,v 1.1 2010/08/13 08:59:28 mg Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::Modules::AgentHelloWorld;

use strict;
use warnings;

use Kernel::System::HelloWorld;

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {%Param};
    bless ($Self, $Type);

    # check needed objects
    for (qw(ParamObject DBObject TicketObject LayoutObject LogObject
QueueObject ConfigObject EncodeObject MainObject)) {
        if ( !$Self->{$_} ) {
            $Self->{LayoutObject}->FatalError( Message => "Got no
$_!" );
        }
    }

    # create needed objects
    $Self->{HelloWorldObject} = Kernel::System::HelloWorld-
>new(%Param);

    return $Self;
}

sub Run {
    my ( $Self, %Param ) = @_;
    my %Data = ();

    $Data{HelloWorldText} = $Self->{HelloWorldObject}-
>GetHelloWorldText();

    # build output
    my $Output = $Self->{LayoutObject}->Header(Title => "HelloWorld");
    $Output .= $Self->{LayoutObject}->NavigationBar();
    $Output .= $Self->{LayoutObject}->Output(
        Data => \%Data,
        TemplateFile => 'AgentHelloWorld',
    );
    $Output .= $Self->{LayoutObject}->Footer();
    return $Output;
}

```

```
1;
```

Core Module

Next, we create the file for the core module `"/HelloWorld/Kernel/System/HelloWorld.pm"` with the following content:

```
# --
# Kernel/System/HelloWorld.pm - core modul
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: writing-otrs-application.xml,v 1.1 2010/08/13 08:59:28 mg Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::HelloWorld;

use strict;
use warnings;

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless ($Self, $Type);

    return $Self;
}

sub GetHelloWorldText {
    my ( $Self, %Param ) = @_;

    return 'Hello World';
}

1;
```

dtl Template File

The last thing missing before the new module can run is the relevant template. Thus, we create the following file:

```
# --
# Kernel/Output/HTML/Standard/AgentHelloWorld.dtl - overview
# Copyright (C) (year) (name of author) (email of author)
```

```

# --
# $Id: writing-otrs-application.xml,v 1.1 2010/08/13 08:59:28 mg Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --
<!-- start form -->
<table border="0" width="100%" cellspacing="0" cellpadding="3">
  <tr>
    <td class="mainhead">
      $Env{"Box0"}$Text{"Overview"}:
      $Text{"HelloWorld"}$Env{"Box1"}
    </td>
  </tr>
  <tr>
    <td class="mainbody">
      <br>
      $Text{"$QData{"HelloWorldText"}"}!<br>
      <br>
      <br>
    </td>
  </tr>
</table>
<!-- end form -->

```

The module is working now and displays the text 'Hello World' when called up.

Language File

If the text 'Hello World' is to be translated into German, a language file for this language must be created: '/HelloWorld/Kernel/Language/de_AgentHelloWorld.pm'. Example:

```

package Kernel::Language::de_AgentHelloWorld;

use strict;
use warnings;

sub Data {
    my $Self = shift;

    $Self->{Translation}->{'Hello World'} = 'Hallo Welt';

    return 1;
}
1;

```

Summary

The example given above shows that it is not too difficult to write a new module for OTRS. It is important, though, to make sure that the module and file name are unique and thus do not

interfere with the framework or other expansion modules. When a module is finished, an OPM package must be generated from it (see chapter 'Package Building').

How to Publish your OTRS Extensions

Package Management

The OPM (OTRS Package Manager) is a mechanism to distribute software packages for the OTRS framework via http, ftp or file upload.

For example, the OTRS project offers OTRS modules like a calendar, a file manager or web mail in OTRS packages via online repositories on our ftp servers. The packages can be managed (install/upgrade/uninstall) via the admin interface.

Package Distribution

If you want to create an OPM online repository, just tell the OTRS framework where the location is. Then you will have a new select option in the admin interface.

```
[Kernel/Config.pm]

# Package::RepositoryList
# (repository list)
$self->{'Package::RepositoryList'} = {
    'ftp://ftp.example.com/packages/' => '[Example-Repository]',
};

[...]
```

Package Repository Index

In your repository, create an index file for your OPM packages. OTRS just reads this index file and knows what packages are available.

```
shell> bin/otrs.PackageManager.pl -a index -d /path/to/repository/ > /
path/to/repository/otrs.xml
shell>
```

Package Commands

You can use the following OPM commands over the admin interface or over bin/otrs.PackageManager.pl to manage admin jobs for OPM packages.

Install

Install OPM packages.

- Web: <http://localhost/otrs/index.pl?Action=AdminPackageManager>
- CMD:

```
shell> bin/otrs.PackageManager.pl -a install -p /path/to/package.opm
```

Uninstall

Uninstall OPM packages.

- Web: <http://localhost/otrs/index.pl?Action=AdminPackageManager>
- CMD:

```
shell> bin/otrs.PackageManager.pl -a uninstall -p /path/to/  
package.opm
```

Upgrade

Upgrade OPM packages.

- Web: <http://localhost/otrs/index.pl?Action=AdminPackageManager>
- CMD:

```
shell> bin/otrs.PackageManager.pl -a upgrade -p /path/to/package.opm
```

List

List all OPM packages.

- Web: <http://localhost/otrs/index.pl?Action=AdminPackageManager>
- CMD:

```
shell> bin/otrs.PackageManager.pl -a list
```

Package Building

If you want to create an OPM package (.opm) you need to create a spec file (.sopm) which includes the properties of the package.

Package Spec File

The OPM package is XML based. You can create/edit the .sopm via a text or xml editor. It contains meta data, a file list and database options.

Name

The package name (required).

```
<Name>Calendar</Name>
```

Version

The package version (required).

```
<Version>1.2.3</Version>
```

Framework

The required framework version (2.4.x means e.g. 2.4.1 or 2.4.9) (required).

```
<Framework>2.4.x</Framework>
```

Can also be used several times.

```
<Framework>2.4.x</Framework>
```

```
<Framework>2.3.x</Framework>
```

```
<Framework>2.2.x</Framework>
```

Vendor

The package vendor (required).

```
<Vendor>OTRS AG</Vendor>
```

URL

The vendor URL (required).

```
<URL>http://otrs.org/</URL>
```

License

The license of the package (required).

```
<License>GNU AFFERO GENERAL PUBLIC LICENSE Version 3, November 2007</License>
```

ChangeLog

The package change log (optional).

```
<ChangeLog Version="1.1.2" Date="2007-02-15 18:45:21">Added some
  feature.</ChangeLog>
<ChangeLog Version="1.1.1" Date="2007-02-15 16:17:51">New package.</
ChangeLog>
```

Description

The package description in different languages (required).

```
<Description Lang="en">A web calendar.</Description>
<Description Lang="de">Ein Web Kalender.</Description>
```

BuildHost

This will be filled in automatically by OPM (auto).

```
<BuildHost>?</BuildHost>
```

BuildDate

This will be filled in automatically by OPM (auto).

```
<BuildDate>?</BuildDate>
```

PackageRequired

Packages that must be installed beforehand (optional). If PackageRequired is used, a version of the required package must be specified.

```
<PackageRequired Version="1.0.3">SomeOtherPackage</PackageRequired>
<PackageRequired Version="5.3.2">SomeotherPackage2</PackageRequired>
```

ModuleRequired

Perl modules that must be installed beforehand (optional).

```
<ModuleRequired Version="1.03">Encode</ModuleRequired>
<ModuleRequired Version="5.32">MIME::Tools</ModuleRequired>
```

OS (^M)

Required OS (optional).

```
<OS>linux</OS>
```

```
<OS>darwin</OS>
<OS>mswin32</OS>
```

Filelist

This is a list of files included in the package (optional).

```
<Filelist>
  <File Permission="644" Location="Kernel/Config/Files/Calendar.pm"/>
>
  <File Permission="644" Location="Kernel/System/CalendarEvent.pm"/>
  <File Permission="644" Location="Kernel/Modules/AgentCalendar.pm"/>
>
  <File Permission="644" Location="Kernel/Language/
de_AgentCalendar.pm"/>
</Filelist>
```

DatabaseInstall

Database entries that have to be created when a package is installed (optional).

```
<DatabaseInstall>
  <TableCreate Name="calendar_event">
    <Column Name="id" Required="true" PrimaryKey="true"
AutoIncrement="true" Type="BIGINT"/>
    <Column Name="title" Required="true" Size="250" Type="VARCHAR"/>
    <Column Name="content" Required="false" Size="250" Type="VARCHAR"/>
  >
  <Column Name="start_time" Required="true" Type="DATE"/>
  <Column Name="end_time" Required="true" Type="DATE"/>
  <Column Name="owner_id" Required="true" Type="INTEGER"/>
  <Column Name="event_status" Required="true" Size="50"
Type="VARCHAR"/>
  </TableCreate>
</DatabaseInstall>
```

You also can choose `<DatabaseInstall Type="post">` or `<DatabaseInstall Type="pre">` to define the time of execution separately (post is default). For more info see chapter "Package Life Cycle".

DatabaseUpgrade

Information on which actions have to be performed in case of an upgrade (subject to version tag), (optional). Example (if already installed package version is below 1.3.4 (e. g. 1.2.6), defined action will be performed):

```
<DatabaseUpgrade>
  <TableCreate Name="calendar_event_involved" Version="1.3.4">
    <Column Name="event_id" Required="true" Type="BIGINT"/>
    <Column Name="user_id" Required="true" Type="INTEGER"/>
  </TableCreate>
</DatabaseUpgrade>
```

```
</TableCreate>
</DatabaseUpgrade>
```

You also can choose `<DatabaseUpgrade Type="post">` or `<DatabaseUpgrade Type="pre">` to define the time of execution separately (post is default). For more info see chapter "Package Life Cycle".

DatabaseReinstall

Information on what actions have to be performed if the package is reinstalled, (optional).

```
<DatabaseReinstall></DatabaseReinstall>
```

You also can choose `<DatabaseReinstall Type="post">` or `<DatabaseReinstall Type="pre">` to define the time of execution separately (post is default). For more info see chapter "Package Life Cycle".

DatabaseUninstall

Uninstall (if a package gets uninstalled), (optional).

```
<DatabaseUninstall>
  <TableDrop Name="calendar_event" />
</DatabaseUninstall>
```

You also can choose `<DatabaseUninstall Type="post">` or `<DatabaseUninstall Type="pre">` to define the time of execution separately (post is default). For more info see chapter "Package Life Cycle".

IntroInstall

To show a "pre" or "post" install introduction in installation dialog.

```
<IntroInstall Type="post" Lang="en" Title="Some Title">
Some Info formatted in dtl/html....
</IntroInstall>
```

You can also use the "Format" attribute to define if you want to use "html" (which is default) or "plain" to use automatically a "`<pre></pre>`" tag wenn intro is shown (to use the new lines and spaces of the content).

IntroUninstall

To show a "pre" or "post" uninstall introduction in uninstallation dialog.

```
<IntroUninstall Type="post" Lang="en" Title="Some Title">
Some Info formatted in dtl/html....
</IntroUninstall>
```

You can also use the "Format" attribute to define if you want to use "html" (which is default) or "plain" to use automatically a "<pre></pre>" tag wenn intro is shown (to use the new lines and spaces of the content).

IntroReinstall

To show a "pre" or "post" reinstall introduction in reinstallation dialog.

```
<IntroReinstall Type="post" Lang="en" Title="Some Title">
Some Info formatted in dtl/html....
</IntroReinstall>
```

You can also use the "Format" attribute to define if you want to use "html" (which is default) or "plain" to use automatically a "<pre></pre>" tag wenn intro is shown (to use the new lines and spaces of the content).

IntroUpgrade

To show a "pre" or "post" upgrade introduction in upgrading dialog.

```
<IntroUpgrade Type="post" Lang="en" Title="Some Title">
Some Info formatted in dtl/html....
</IntroUpgrade>
```

You can also use the "Format" attribute to define if you want to use "html" (which is default) or "plain" to use automatically a "<pre></pre>" tag wenn intro is shown (to use the new lines and spaces of the content).

CodeInstall

To execute perl code if the package is installed (optional).

```
<CodeInstall>
# example
if (1) {
    print STDERR "Some info to STDERR\n";
}
# log example
$self->{LogObject}->Log(
    Priority => 'notice',
    Message => "Some Message!",
)
# database example
$self->{DBObject}->Do(SQL => "SOME SQL");
</CodeInstall>
```

You also can choose <CodeInstall Type="post"> or <CodeInstall Type="pre"> to define the time of execution separately (post is default). For more info see chapter "Package Life Cycle".

CodeUninstall

To execute perl code if the package is uninstalled (optional). On "pre" or "post" time of package uninstallation.

```
<CodeUninstall>
  # example
  if (1) {
    print STDERR "Some info to STDERR\n";
  }
</CodeUninstall>
```

You also can choose `<CodeUninstall Type="post">` or `<CodeUninstall Type="pre">` to define the time of execution separately (post is default). For more info see chapter "Package Life Cycle".

CodeReinstall

To execute perl code if the package is reinstalled (optional).

```
<CodeReinstall>
  # example
  if (1) {
    print STDERR "Some info to STDERR\n";
  }
</CodeReinstall>
```

You also can choose `<CodeReinstall Type="post">` or `<CodeReinstall Type="pre">` to define the time of execution separately (post is default). For more info see chapter "Package Life Cycle".

CodeUpgrade

To execute perl code if the package is upgraded (subject to version tag), (optional). Example (if already installed package version is below 1.3.4 (e. g. 1.2.6), defined action will be performed):

```
<CodeUpgrade Version="1.3.4">
  # example
  if (1) {
    print STDERR "Some info to STDERR\n";
  }
</CodeUpgrade>
```

You also can choose `<CodeUpgrade Type="post">` or `<CodeUpgrade Type="pre">` to define the time of execution separately (post is default).

Example .sopm

This is a whole example spec file.

```
<?xml version="1.0" encoding="utf-8" ?>
```

```

<otrs_package version="1.0">
  <Name>Calendar</Name>
  <Version>0.0.1</Version>
  <Framework>2.4.x</Framework>
  <Vendor>OTRS AG</Vendor>
  <URL>http://otrs.org/</URL>
  <License>GNU GENERAL PUBLIC LICENSE Version 2, June 1991</License>
  <ChangeLog Version="1.1.2" Date="2007-02-15 18:45:21">Added some
feature.</ChangeLog>
  <ChangeLog Version="1.1.1" Date="2007-02-15 16:17:51">New
package.</ChangeLog>
  <Description Lang="en">A web calendar.</Description>
  <Description Lang="de">Ein Web Kalender.</Description>
  <IntroInstall Type="post" Lang="en" Title="Thank you!">Thank you
for choosing the Calendar module.</IntroInstall>
  <IntroInstall Type="post" Lang="de" Title="Vielen Dank!">Vielen
Dank fuer die Auswahl des Kalender Modules.</IntroInstall>
  <BuildDate>?</BuildDate>
  <BuildHost>?</BuildHost>
  <Filelist>
    <File Permission="644" Location="Kernel/Config/Files/
Calendar.pm"></File>
    <File Permission="644" Location="Kernel/System/
CalendarEvent.pm"></File>
    <File Permission="644" Location="Kernel/Modules/
AgentCalendar.pm"></File>
    <File Permission="644" Location="Kernel/Language/
de_AgentCalendar.pm"></File>
    <File Permission="644" Location="Kernel/Output/HTML/Standard/
AgentCalendar.dtl"></File>
    <File Permission="644" Location="Kernel/Output/HTML/
NotificationCalendar.pm"></File>
    <File Permission="644" Location="var/httpd/htdocs/images/
Standard/calendar.png"></File>
  </Filelist>
  <DatabaseInstall>
    <TableCreate Name="calendar_event">
      <Column Name="id" Required="true" PrimaryKey="true"
AutoIncrement="true" Type="BIGINT"/>
      <Column Name="title" Required="true" Size="250"
Type="VARCHAR"/>
      <Column Name="content" Required="false" Size="250"
Type="VARCHAR"/>
      <Column Name="start_time" Required="true" Type="DATE"/>
      <Column Name="end_time" Required="true" Type="DATE"/>
      <Column Name="owner_id" Required="true" Type="INTEGER"/>
      <Column Name="event_status" Required="true" Size="50"
Type="VARCHAR"/>
    </TableCreate>
  </DatabaseInstall>
  <DatabaseUninstall>
    <TableDrop Name="calendar_event"/>
  </DatabaseUninstall>
</otrs_package>

```

Package Build

To build an .opm package from the spec opm.

```
shell> bin/otrs.PackageManager.pl -a build -p /path/to/example.sopm
writing /tmp/example-0.0.1.opm
shell>
```

Package Life Cycle - Install/Upgrade/Uninstall

The following image shows you how the life cycle of a package installation/upgrade/uninstallation works in the backend step by step.

How to Upgrade your OTRS Extensions to Newer Versions of OTRS

Usually, new major OTRS releases also bring changes in the internals which need to be reflected in extension modules too. This chapter will describe these changes for each version upgrade.

Upgrading OTRS Extensions from 2.4 to 3.0

Translation of configuration file descriptions.

Previously, OTRS had translations for items in the sysconfig in multiple language in the XML files. Starting OTRS 3.0.x, the translations are removed from the configuration file. The configuration file only stores the English descriptions. If needed, you can localize the descriptions in your translation file in the way 'regular' strings in the web interface are translated. This has the added benefit that if a translator wants to localize a module's configuration, this can be fully done from the translation file. The `otrs.CreateTranslationFile.pl` script has been adjusted to also grab strings from the SysConfig.

Additionally, if you have tags that are configured in SysConfig but displayed in the GUI, you can indicate with the option `Translatable="1"` that this is a field that should be added to the translation file.

```
<ConfigItem Name="ProductName" Required="1" Valid="1">
  <Description Lang="en">This setting controls the name of the
application as is shown in the web interface as well as the tabs and
title bar of your web browser.</Description>
  <Description Lang="de">Im WebFrontend angezeigter Name der
Software.</Description>
  <Group>Framework</Group>
  <SubGroup>Core</SubGroup>
  <Setting>
    <String Regex="">OTRS</String>
  </Setting>
</ConfigItem>
```

New Style:

```
<ConfigItem Name="ProductName" Required="1" Valid="1"
ConfigLevel="200">
  <Description Translatable="1">Defines the name of the
application, shown in the web interface, tabs and title bar of the
web browser.</Description>
  <Group>Framework</Group>
  <SubGroup>Core</SubGroup>
  <Setting>
    <String Regex="">OTRS</String>
  </Setting>
</ConfigItem>
```

Please note that the `Block` names changed (`Agent`, `Customer`, `Email`, `Queue`, `Ticket` and `System` are available). An additional `Description` tag has been added.

Configuration of `NavBarModule` was Changed

Old Style:

```
<ConfigItem Name="Frontend::Module###AdminUser" Required="0"
Valid="1">
  <Description Lang="en">Frontend module registration for the
AdminUser object in the admin area.</Description>
  <Description Lang="de">Frontendmodul-Registrierung des AdminUser-
Objekts im Admin-Bereich.</Description>
  <Group>Framework</Group>
  <SubGroup>Frontend::Admin::ModuleRegistration</SubGroup>
  <Setting>
    <FrontendModuleReg>
      <Group>admin</Group>
      <Description>Admin</Description>
      <Title>User</Title>
      <NavBarName>Admin</NavBarName>
      <NavBarModule>
        <Module>Kernel::Output::HTML::NavBarModuleAdmin</
Module>
        <Name>Users</Name>
        <Block>Block1</Block>
        <Prio>100</Prio>
      </NavBarModule>
    </FrontendModuleReg>
  </Setting>
</ConfigItem>
```

New Style:

```

    <ConfigItem Name="Frontend::Module###AdminUser" Required="0"
Valid="1">
    <Description Translatable="1">Frontend module registration for
the agent interface.</Description>
    <Group>Framework</Group>
    <SubGroup>Frontend::Admin::ModuleRegistration</SubGroup>
    <Setting>
    <FrontendModuleReg>
    <Group>admin</Group>
    <Description>Create and manage agents.</Description>
    <Title>Agents</Title>
    <NavBarName>Admin</NavBarName>
    <NavBarModule>
    <Module>Kernel::Output::HTML::NavBarModuleAdmin</
Module>
    <Name Translatable="1">Agents</Name>
    <Description Translatable="1">Create and manage
agents.</Description>
    <Block>Agent</Block>
    <Prio>100</Prio>
    </NavBarModule>
    </FrontendModuleReg>
    </Setting>
</ConfigItem>

```

Please note that the `Block` names changed (Agent, Customer, Email, Queue, Ticket and System are available). An additional `Description` tag has been added.

Configuration of NavBar was Changed

Old Style:

```

    <ConfigItem Name="Frontend::Module###AgentTicketSearch"
Required="0" Valid="1">
    <Description Lang="en">Frontend module registration for the
AgentTicketSearch object in the agent interface.</Description>
    <Description Lang="de">Frontendmodul-Registration des
AgentTicketSearch-Objekts im Agent-Interface.</Description>
    <Group>Ticket</Group>
    <SubGroup>Frontend::Agent::ModuleRegistration</SubGroup>
    <Setting>
    <FrontendModuleReg>
    <Description>Search Tickets</Description>
    <Title>Search</Title>
    <NavBarName>Ticket</NavBarName>
    <NavBar>
    <Description>Search Tickets</Description>
    <Name>Search</Name>
    <Image>search.png</Image>
    <Link>Action=AgentTicketSearch</Link>
    <NavBar>Ticket</NavBar>
    <Type></Type>

```

```

        <Block></Block>
        <AccessKey>s</AccessKey>
        <Prio>300</Prio>
    </NavBar>
</FrontendModuleReg>
</Setting>
</ConfigItem>

```

New Style:

```

    <ConfigItem Name="Frontend::Module###AgentTicketSearch"
    Required="0" Valid="1">
        <Description Translatable="1">Frontend module registration for
the agent interface.</Description>
        <Group>Ticket</Group>
        <SubGroup>Frontend::Agent::ModuleRegistration</SubGroup>
        <Setting>
            <FrontendModuleReg>
                <Description>Search Ticket</Description>
                <Title>Search</Title>
                <NavBarName>Ticket</NavBarName>
                <NavBar>
                    <Description Translatable="1">Search Ticket</
Description>
                    <Name Translatable="1">Search</Name>
                    <Link>Action=AgentTicketSearch</Link>
                    <LinkOption></LinkOption>
                    <NavBar>Ticket</NavBar>
                    <Type></Type>
                    <Block></Block>
                    <AccessKey>s</AccessKey>
                    <Prio>300</Prio>
                </NavBar>
            </FrontendModuleReg>
        </Setting>
    </ConfigItem>

```

Please note that the `NavBar` names changed (Description Name Link LinkOption NavBar Type Block AccessKey Prio are available). An additional `LinkOption` tag has been added and `Image` tag has been removed.

Configuration Setting Frontend::NavBarModule was Renamed

Frontend::NavBarModule was renamed to Frontend::ToolBarModule. Block option need to get replaced, ToolBarItem is now used.

PreferencesGroups configuration

Up to this point, the PreferencesGroups (which can be used to create a section in the Agent or Customer configuration dialog) had either a description (`Desc`), or a field label (`Key`), or both. With the new GUI, the interface was slightly changed so that now the `Key` always must be provided.

The existing entries were changed so that either the `Desc` was shortened and turned into a `Key` instead, or a concise `Key` was added to describe the setting in Question.

This is also what should be done (either of the two options) when porting a custom `PreferencesGroups` module to OTRS 3.0.

Example:

```
<ConfigItem Name="PreferencesGroups###RefreshTime" Required="0"
Valid="1">
  <Description Translatable="1">Parameters for the RefreshTime
object in the preference view of the agent interface.</Description>
  <Group>Ticket</Group>
  <SubGroup>Frontend::Agent::Preferences</SubGroup>
  <Setting>
    <Hash>
      <Item
Key="Module">Kernel::Output::HTML::PreferencesGeneric</Item>
      <Item Key="Column">Other Settings</Item>
      <Item Key="Label" Translatable="1">QueueView Refresh
Time</Item>
      <Item Key="Desc" Translatable="1">If enabled, the
QueueView will automatically refresh after the specified time.</Item>
      <Item Key="Key" Translatable="1">Refresh QueueView
after</Item>
      <Item Key="Data">
        <Hash>
          <Item Key="0">off</Item>
          <Item Key="2"> 2 minutes</Item>
          <Item Key="5"> 5 minutes</Item>
          <Item Key="7"> 7 minutes</Item>
          <Item Key="10">10 minutes</Item>
          <Item Key="15">15 minutes</Item>
        </Hash>
      </Item>
      <Item Key="DataSelected">0</Item>
      <Item Key="PrefKey">UserRefreshTime</Item>
      <Item Key="Prio">2000</Item>
      <Item Key="Active">1</Item>
    </Hash>
  </Setting>
</ConfigItem>
```

`Key` is mandatory, `Desc` is optional.

Chapter 3. Contributing to OTRS

This chapter will show how you can contribute to the OTRS framework, so that other users will be able to benefit from your work.

Translating OTRS

The OTRS framework allows for different languages to be used in the frontend.

How it works

There are three different translation file types which are used in the following order. If a word/sentence is redefined in a translation file, the latest definition will be used.

1. Default Framework Translation File

```
Kernel/Language/$Language.pm
```

2. Frontend Module Translation File

```
Kernel/Language/$Language_$FrontendModule.pm
```

3. Custom Translation File

```
Kernel/Language/$Language_Custom.pm
```

Default Framework Translation File

The Default Framework Translation File includes the basic translations. The following is an example of a Default Framework Translation File.

Format:

```
package Kernel::Language::de;

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.3 $) [1];

sub Data {
    my $Self = shift;

    # $$START$$

    # possible charsets
    $Self->{Charset} = ['iso-8859-1', 'iso-8859-15', ];
    # date formats (%A=WeekDay;%B=LongMonth;%T=Time;%D=Day;%M=Month;
    %Y=Year;)
```

```

$self->{DateFormat} = '%D.%M.%Y %T';
$self->{DateFormatLong} = '%A %D %B %T %Y';
$self->{DateFormatShort} = '%D.%M.%Y';
$self->{DateInputFormat} = '%D.%M.%Y';
$self->{DateInputFormatLong} = '%D.%M.%Y - %T';

$self->{Translation} = {
# Template: AAABase
'Yes' => 'Ja',
'No' => 'Nein',
'yes' => 'ja',
'no' => 'kein',
'Off' => 'Aus',
'off' => 'aus', Kernel/Language/$Language_Custome.pm
};
# $$STOP$$
return 1;
}

1;

```

Frontend Translation File

The Frontend Translation File is used for a specific frontend module. If, for example, the frontend module "Kernel/Modules/AgentCalendar.pm", also <http://otrs.example.com/otrs/index.pl?Action=AgentCalendar>, is used, the Frontend Translation File "Kernel/Language/de_Agentcalendar.pm" is used, too.

Format:

```

package Kernel::Language::de_AgentCalendar;

use strict;
use warnings;

sub Data {
    my $self = shift;

    $self->{Translation}->{'CW'} = 'KW';
    $self->{Translation}->{'Today'} = 'heute';
    $self->{Translation}->{'Tomorrow'} = 'Morgen';
    $self->{Translation}->{'1 St. May'} = 'Erster Mai';
    $self->{Translation}->{'Christmas'} = 'Weihnachten';
    $self->{Translation}->{'Silvester'} = 'Silvester';
    $self->{Translation}->{'New Year\'s Eve!'} = 'Neu Jahr!';
    $self->{Translation}->{'January'} = 'Januar';
    $self->{Translation}->{'February'} = 'Februar';
    return 1;
}

1;

```

Custom Translation File

The Custom Translation File is read out last and so its translation which will be used. If you want to add your own wording to your installation, create this file for your language.

Format:

```
package Kernel::Language::xx_Custom;

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.3 $) [1];

sub Data {
    my $Self = shift;

    # $$START$$

    # own translations
    $Self->{Translation}->{'Lock'} = 'Lala';
    $Self->{Translation}->{'Unlock'} = 'Lulu';

    # $$STOP$$
    return 1;
}

1;
```

Updating an existing translation

Updating an existing translation is easy:

1. To coordinate translation efforts, avoiding duplicate translation efforts, and to send in your translated files, please use the i18n mailing list [<http://lists.otrs.org/cgi-bin/listinfo/i18n>] of OTRS.
2. Take the current translation file (Kernel/Language/\$Language.pm) from CVS (<http://source.otrs.org> [<http://source.otrs.org/>]) and save it in your Kernel/Language/ directory, overwriting the current one in your filesystem.
3. Now you can update the file.
4. When you are finished and satisfied with the translation, please send it to the i18n mailing list.

Adding a new frontend translation

If you want to translate the OTRS framework into a new language, you have to follow these steps:

1. To coordinate translation efforts, avoiding duplicate translation efforts, and to send in your translated files, please use the i18n mailing list [<http://lists.otrs.org/cgi-bin/listinfo/i18n>] of OTRS.

2. Take the current German translation (Kernel/Language/de.pm) from CVS (<http://source.otrs.org> [<http://source.otrs.org/>]). Use the German version because this is always up to date.
3. Change the package name (e.g. "package Kernel::Language::de;" to "package Kernel::Language::fr;") and translate each word/sentence.
4. Add the new language translation to the framework by adding it to your `Kernel/Config.pm`.

```
$Self->{DefaultUsedLanguages}->{fr} = 'French';
```

5. If you use `mod_perl`, restart your webserver and the new language will be shown in your preferences selection.
6. When you are finished and satisfied with the translation, please send it to the i18n mailing list.

Translating the Documentation

The OTRS admin manual can be translated. The workflow is like this:

1. The original documentation is written in English, in docbook XML format.
2. The tool `po4a` [<http://po4a.alioth.debian.org/>] extracts gettext translation files from that (.po). These files can be edited with existing helpful tools like Poedit [<http://www.poedit.net/>].
3. These files can be translated and will be updated on changes in the English source files without losing existing translations.
4. From the translation files, translated docbook XML files can be generated for the different languages (again with `po4a`).

This process has the benefit of keeping track of updates in the original source files. Only additions and updates have to be retranslated.

The CVS repository for the admin documentation can be found here [<http://source.otrs.org/viewvc.cgi/doc-admin/>]. Inside of this repository, there is the directory with the translation files [<http://source.otrs.org/viewvc.cgi/doc-admin/i18n/>].

To start a new translation, download the translation template file `doc-admin.pot` [<http://source.otrs.org/viewvc.cgi/doc-admin/i18n/doc-admin.pot?view=co>] from CVS, save it and rename the file to `doc-admin.$Language.po` (e. g. `doc-admin.de.po` for German).

To improve an existing translation, you can download the `doc-admin.$Language.po` file for this language and work on this. If there are questions on how the translation should be made, you can use `doc-admin.de.po` as a reference.

It is important that the structure of the generated XML stays intact. So if the original string is "Edit <filename>Kernel/Config.pm</filename>", then the German translation has to be "<filename>Kernel/Config.pm</filename> bearbeiten", keeping the the XML tags. Scripts and examples usually do not have to be translated (so you can just copy the source text to the translation text field in this case).

To coordinate translation efforts and to send in your translated .po files, please use the i18n mailing list [<http://lists.otrs.org/cgi-bin/listinfo/i18n>] of OTRS. The OTRS team will take care of regenerating the XML documentation for your language.

Important

Especially if you add a new translation, it is recommended to send in snapshots of your file regularly to make sure that your translation has the correct structure and that the docbook documentation can be successfully generated from it, producing valid XML documents.

If you want to test your translation file locally, you can checkout the doc-admin repository and use po4a directly to generate the translation files. Please see the file `po4a.conf` in the top directory of that module for usage hints. For additional questions, please use the i18n mailing list.

Code Style Guide

In order to preserve the consistent development of the OTRS project, we have set up guidelines regarding style for the different programming languages.

Perl

Formatting

TAB: We use 4 spaces. Examples for braces:

```
if ($Condition) {
    Foo();
}
else {
    Bar();
}

while ($Condition == 1) {
    Foo();
}
```

Naming

Names and comments are written in English. Variables, Objects and Methods must be descriptive nouns or noun phrases with the first letter set upper case.

e. g. `@TicketIDs` or `$Output` or `BuildQueueView()`

Source Code Header and Charset

Attach the following header to each and every source file. Source files are saved in Charset ISO-8859-1.

```
# --
# (file name) - a short description what it does
```

```
# Copyright (C) 2001-2011 OTRS AG, http://otrs.org/
# --
# $Id: perl.xml,v 1.3 2011/03/09 14:54:40 mb Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --
```

Executable files (*.pl) have a special header.

```
#!/usr/bin/perl -w
# --
# (file name) - a short description what it does
# Copyright (C) 2001-2011 OTRS AG, http://otrs.org/
# --
# $Id: perl.xml,v 1.3 2011/03/09 14:54:40 mb Exp $
# --
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU AFFERO General Public License as
# published by
# the Free Software Foundation; either version 3 of the License, or
# any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU Affero General Public
# License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
# USA
# or see http://www.gnu.org/licenses/agpl.txt.
# --
```

The following line is updated by the CVS:

```
# $Id: perl.xml,v 1.3 2011/03/09 14:54:40 mb Exp $
```

Version Comments

Some functions may not be available in the current framework version. Thus, sometimes settings have to be changed when a new module and framework version will be released. If you use version comments, you can search for information (e.g. using `grep`) on what must be changed when a new version is published. The keyword for version comments is 'FRAMEWORK' For perl use '#' and for xml use the xml comments.

```
e.g. for perl-code
# FRAMEWORK-2.1: the function ID2UserName is first available in OTRS
2.1
```

Special comments

The only ways to create special comments are the following ways. Example 1 - especially for subactions of frontend modules

```
# -----#
# here starts a special area
# -----#
```

Example 2 - especially for customizing standard OTRS files

```
# --- customizing for bsi
```

Restrictions for some functions

Some functions are not useful in every script. Please pay attention to the following restrictions.

- don't use "die" and "exit" in .pm-files
- don't use the "Dumper" function in released files
- don't use "print" in .pm files
- use OTRS specific function "SystemTime2Date" instead of "localtime"

Perldoc

Every function which could be used outside of its package must have a perldoc. It should look like the following example.

```
=item SystemTime2TimeStamp()

returns a time stamp in "yyyy-mm-dd 23:59:59" format.

    my $TimeStamp = $TimeObject->SystemTime2TimeStamp(
        SystemTime => $SystemTime,
    );
```

If you need the short format "23:59:59" for dates that are "today", pass the Type parameter like this:

```
my $TimeStamp = $TimeObject->SystemTime2TimeStamp(
    SystemTime => $SystemTime,
    Type       => 'Short',
```

```
);  
=cut
```

Length of lines

Please see that a line of code is not longer than 100 characters.

Core-Objects

Objects and their allocation

In OTRS many objects are available. But it is not allowed to use every object in each script. Please note the following definitions

- don't use the LayoutObject in core modules
- don't use the ParamObject in core modules
- don't use the DBObject in frontend modules

Using of the MainObject

Information about the MainObject

- initialize the MainObject in the basic .pl-file
- in .pm files only pass it to the next Object you initialize
- don't use the Perl "require" function any more

Using of the EncodeObject

Information about the EncodeObject

- initialize the EncodeObject in the basic .pl-file
- in .pm files only pass it to the next Object you initialize

Regular Expressions

For regular expressions we always use the `m//` operator with curly braces as delimiters. We also use the modifiers `x`, `m` and `s`. The `x` modifiers allows you to comment your regex and use spaces to "group" logical groups.

```
$Date =~ m{ \A \d{4} - \d{2} - \d{2} \z }xms  
$Date =~ m{  
    \A      # beginning of the string  
    \d{4} - # year  
    \d{2} - # month  
    [^\n]  # everything but newline  
    #..
```

```
}xms;
```

As the space has no longer a special meaning, you have to use a single character class to match a single space ([]). If you want to match any whitespace you can use \s. In the regex, the dot ('.') includes the newline (whereas in regex without s modifier the dot means 'everything but newline'). If you want to match anything but newline, you have to use the negated single character class ([^\n]).

```
$Text =~ m{
    Test
    [ ]    # there must be a space between 'Test' and 'Regex'
    Regex
}xms;
```

JavaScript

Browser Handling

All JavaScript is loaded in all browsers (no browser hacks in the .dtl files). The code is responsible to decide if it has to skip or execute certain parts of itself only in certain browsers.

Directory Structure

Directory structure inside the js/ folder:

```
* js
  * thirdparty          # thirdparty libs always have the
  version number inside the directory
  * ckeditor-3.0.1
  * jquery-1.3.2
  * Core.Agent.*       # stuff specific to the agent interface
  * Core.Customer.*   # customer interface
  * Core.*             # common API
```

Thirdparty Code

Every thirdparty module gets its own subdirectory: "module name"- "version number" (e.g. ckeditor-3.0.1, jquery-1.3.2). Inside of that, file names should not have a version number or postfix included (wrong: jquery/jquery-1.4.3.min.js, right: jquery-1.4.3/jquery.js).

Variables

- Variable names should be CamelCase, just like in Perl.
- Variables that hold a jQuery object should start with \$, for example: \$Tooltip.

Functions

- Function names should be !CamelCase, just like in Perl.

Namespaces

- TODO...

Comments

- Single line comments are done with `//`.
- Longer comments are done with `/* ... */`
- If you comment out parts of your JavaScript code, only use `//` because `/* .. */` can cause problems with Regular Expressions in the code.

Event Handling

- Always use `$.bind()` instead of the event-shorthand methods of jQuery for better readability (wrong: `$SomeObject.click(...)`, right: `$SomeObject.bind('click', ...)`).
- Do not use `$.live()`! We had severe performance issues with `$.live()` in correlation with mouse events. Until it can be verified that `$.live()` works with other event types without problems.
- If you `$.bind()` events, make sure to `$.unbind()` them beforehand, to make sure that events will not be bound twice, should the code be executed another time.

CSS

- Minimum resolution is 1024x768px.
- The layout is liquid, which means that if the screen is wider, the space will be used.
- Absolute size measurements should be specified in px to have a consistent look on many platforms and browsers.
- Documentation is made with CSSDOC (see CSS files for examples). All logical blocks should have a CSSDOC comment.

Architecture

- We follow the Object Oriented CSS [<http://wiki.github.com/stubbornella/occss/>] approach. In essence, this means that the layout is achieved by combining different generic building blocks to realize a particular design.
- Wherever possible, module specific design should not be used. Therefore we also do not work with IDs on the body element, for example, if it can be avoided.

Style

- All definitions have a `{` in the same line as the selector, all rules are defined in one row per rule, the definition ends with a row with a single `}` in it. See the following example:

```
#Selector {
  width: 10px;
  height: 20px;
```

```
padding: 4px;
}
```

- Between `:` and the rule value, there is a space
- Every rule has an indent of 4 spaces.
- If multiple selectors are specified, separate them with comma and put each one on an own line:

```
#Selector1,
#Selector2,
#Selector3 {
    width: 10px;
}
```

- If rules are combinable, combine them (e.g. combine `background-position`, `background-image`, ... into `background`).
- Rules should be in a logical order within a definition (all color specific rule together, all positioning rules together, ...).
- All IDs and Names are written in CamelCase notation:

```
<div class="NavigationBar" id="AdminMenu"></div>
```

User Interface Design

Capitalization

This section talks about how the different parts of the English user interface should be capitalized. For further information, you may want to review this helpful page [<http://msdn.microsoft.com/en-us/library/aa974176.aspx#capitalization>].

- Headings (h1-h6) and Titles (Names, such as `Queue View`) are set in "title style" capitalization, that means all first letters will be capitalized (with a few exceptions such as "this", "and", "or" etc.).

Examples: `Action List`, `Manage Customer-Group Relations`.

- Other structural elements such as buttons, labels, tabs, menu items are set in "sentence style" capitalization (only the first letter of a phrase is capitalized), but no final dot is added to complete the phrase as a sentence.

Examples: `First name`, `Select queue refresh time`, `Print this ticket`.

- Descriptive texts and tooltip contents are written as complete sentences.

Example: `This value is required`.

- For translations, it has to be checked if the title style capitalization is also appropriate in the target language, it might have to be changed to sentence style capitalization or something else.

Accessibility Guide

This document is supposed to explain basics about accessibility issues and give guidelines for contributions to OTRS.

Accessibility Basics

What is Accessibility?

Accessibility is a general term used to describe the degree to which a product, device, service, or environment is accessible by as many people as possible. Accessibility can be viewed as the "ability to access" and possible benefit of some system or entity. Accessibility is often used to focus on people with disabilities and their right of access to entities, often through use of assistive technology.

In the context of web development, accessibility has a focus on enabling people with impariments full access to web interfaces. For example, this group of people can include partially visually impaired or completely blind people. While the former can still partially use the GUI, the latter have to completely rely on assistive technologies such as software which reads the screen to them (screen readers).

Why is it important for OTRS?

To enable impaired users access to OTRS systems is a valid goal in itself. It shows respect.

Furthermore, fulfilling accessibility standards is becoming increasingly important in the public sector (government institutions) and large companies, which both belong to the target markets of OTRS.

How can I successfully work on accessibility issues even if I am not disabled?

This is very simple. Pretend to be blind.

Don't

- use the Mouse and
- look at the screen.

Then try to use OTRS with the help of a screen reader and your keyboard only. This should give you an idea of how it will feel for a blind person.

Ok, but I don't have a screen reader!

While commercial screen readers such as JAWS (perhaps the best known one) can be extremely expensive, there are OpenSource screen readers which you can install and use:

- NVDA [<http://www.nvda-project.org/>], a screen reader for Windows. (Use the 2010.beta1 or any later version as this has better support for the web accessibility standards).
- ORCA [<http://live.gnome.org/Orca>], a screen reader for Gnome/Linux.

Now you don't have an excuse any more. ;)

Accessibility Standards

This section is included for reference only, you do not have to study the standards themselves to be able to work on accessibility issues in OTRS. We'll try to extract the relevant guidelines in this document.

Web Content Accessibility Guidelines (WCAG)

This W3C standard gives general guidelines for how to create accessible web pages.

- WCAG 2.0 [<http://www.w3.org/TR/WCAG20/>]
- How to Meet WCAG 2.0 [<http://www.w3.org/WAI/WCAG20/quickref/>]
- Understanding WCAG 2.0 [<http://www.w3.org/TR/UNDERSTANDING-WCAG20/>]

WCAG has different levels of accessibility support. We currently plan to support level A, as AA and AAA deal with matters that seem not relevant for OTRS.

Accessible Rich Internet Applications (WAI-ARIA) 1.0

This standard (ist is still in draft status, in fact) deals with the special issues arising from the shift away from static content to dynamic web applications. It deals with questions like how a user can be notified of changes in the user interface resulting from AJAX requests, for example.

- WAI-ARIA 1.0 [<http://www.w3.org/TR/wai-aria/>]

Implementation guidelines

Provide alternatives for non-text content

Goal: All non-text content that is presented to the user has a text alternative that serves the equivalent purpose (WCAG 1.1.1)

It is very important to understand that screen readers can only present textual information and available metadata to the user. To give you an example, whenever a screen reader sees ``, it can only read "link" to the user, but not the purpose of this link. With a slight improvement, it would be accessible: ``. In this case the user would hear "link close this widget", voila!

It is important to always formulate the text in a most "speaking" way. Just imagine it is the only information that you have. Will it help you? Can you understand its purpose just by hearing it?

Please follow these rules when working on OTRS:

- *Rule:* Wherever possible, use speaking texts and formulate in real, understandable and precise sentences. "Close this widget" is much better than "Close", because the latter is redundant.
- *Rule:* Links always must have either text content that is spoken by the screen reader (`Delete this entry`), or a title attribute (``).
- *Rule:* Images must always have an alternative text that can be read to the user (``).

Make navigation easy

Goal: *allow the user to easily navigate the current page and the entire application.*

The `title` tag is the first thing a user hears from the screen reader when opening a web page. For OTRS, there is also always just one `h1` element on the page, indicating the current page (it contains part of the information from `title`). This navigational information helps the user to understand where they are, and what the purpose of the current page is.

- *Rule:* Always give a precise title to the page that allows the user to understand where they currently are.

Screen readers can use the built-in document structure of HTML (headings `h1` to `h6`) to determine the structure of a document and to allow the user to jump around from section to section. However, this is not enough to reflect the structure of a dynamic web application. That's why ARIA defines several "landmark" roles that can be given to elements to indicate their navigational significance. To keep the validity of the HTML documents, the role attributes (ARIA landmark roles) are not inserted into the source code directly, but instead by classes which will later be used by the JavaScript functions in `OTRS.UI.Accessibility` to set the corresponding role attributes on the node.

- *Rule:* Use WAI ARIA Landmark Roles to structure the content for screenreaders
 - **Banner:** `<div class="ARIARoleBanner"></div>` will become `<div class="ARIARoleBanner" role="banner"></div>`
 - **Navigation:** `<div class="ARIARoleNavigation"></div>` will become `<div class="AriaRoleNavigation" role="navigation"></div>`
 - **Search function:** `<div class="ARIARoleSearch"></div>` will become `<div class="ARIARoleSearch" role="search"></div>`
 - **Main application area:** `<div class="ARIARoleMain"></div>` will become `<div class="ARIARoleMain" role="main"></div>`
 - **Footer:** `<div class="ARIARoleContentinfo"></div>` will become `<div class="ARIARoleContentinfo" role="contentinfo"></div>`

For navigation inside of `<form>` elements, it is necessary for the impaired user to know what each input elements purpose is. This can be achieved by using standard HTML `<label>` elements which create a link between the label and the form element. When an input element gets focus, the screen reader will usually read the connected label, so that the agent can hear its exact purpose. An additional benefit for seeing users is that they can click on the label, and the input element will get focus (especially helpful for checkboxes, for example).

- *Rule:* Provide `<label>` elements for **all** form element (input, select, textarea) fields.

Example: `<label for="date">Date:</label><input type="text" name="date" id="date"/>`

Make interaction possible

Goal: *Allow the user to perform all interactions just by using the keyboard.*

While it is technically possible to create interactions with JavaScript on arbitrary HTML elements, this must be limited to elements that a user can interact with by using the keyboard. Specifically,

they need to be able to give focus to the element and to interact with it. For example, a push button to toggle a widget should not be realized by using a `span` element with an attached JavaScript `onclick` event listener, but it should be (or contain) an `a` tag to make it clear to the screen reader that this element can cause interaction.

- *Rule:* For interactions, always use elements that can receive focus, such as `a`, `input`, `select` and `button`.
- *Rule:* Make sure that the user can always identify the nature of the interaction (see rules about non-textual content and labelling of form elements).

Goal: Make dynamic changes known to the user.

A special area of accessibility problems are dynamic changes in the user interface, either by JavaScript or also by AJAX calls. The screen reader will not tell the user about changes without special precautions. This is a difficult topic and cannot yet be completely explained here.

- *Rule:* Always use the validation framework `OTRS.Validate` for form validation.

This will make sure that the error tooltips are being read by the screen reader. That way the blind agent a) knows the item which has an error and b) get a text describing the error.

- *Rule:* Use the function `OTRS.UI.Accessibility.AudibleAlert()` to notify the user about other important UI changes.
- *Rule:* Use the `OTRS.UI.Dialog` framework to create modal dialogs. These are already optimized for accessibility.

General screen reader optimizations

Goal: Help screen readers with their work.

- *Rule:* Each page must identify its own main language so that the screenreader can choose the right speech synthesis engine.

Example: `<html lang="fr">...</html>`

Unit Tests

OTRS provides unit tests for core modules.

Creating a test file

The test files are stored in `.t` files under `/scripts/test/*t`. For example the file `/scripts/test/Calendar.t` for the Calendar Module.

A test file consists of the function call of the function to be tested and the analysis of the return value. Example (`/scripts/test/Calendar.t`):

```
# --
# Calendar.t - Calendar
# Copyright (C) 2001-2010 OTRS AG, http://otrs.org/
# --
# $Id: unit-tests.xml,v 1.2 2010/08/26 14:59:57 mb Exp $
```

```

# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

use strict;
use warnings;
use utf8;

use vars qw($Self);

use Kernel::System::User;
use Kernel::System::CalendarEvent;

$Self->{UserObject} = Kernel::System::User->new(%{$Self});
$Self->{EventObject} = Kernel::System::CalendarEvent->new(%{$Self},
    UserID => 1);

my $EventID = $Self->{EventObject}->EventAdd(
    Title => 'Some Test',
    StartTime => '1977-10-27 20:15',
    EndTime => '1977-10-27 21:00',
    State => 'public',
    UserIDs => [1],
);

$Self->True(
    $EventID,
    'EventAdd()',
);

[...]
```

Testing

To check your tests, just use "bin/otrs.UnitTest.pl -n Calendar" to use /scripts/test/Calendar.t.

```

shell:/opt/otrs> bin/otrs.UnitTest.pl -n Calendar
+-----+
/opt/otrs/scripts/test/Calendar.t:
+-----+
ok 1 - EventAdd()
=====
Product:   OTRS 3.0.x CVS
Test Time: 0 s
Time:      2010-04-02 12:58:37
Host:      yourhost.example.com
Perl:      5.8.9
OS:        linux
TestOk:    1
```

```
TestNotOk: 0
=====
shell:/opt/otrs>
```

True()

This function tests whether the return value of the function 'EventAdd()' in the variable \$EventID is valid.

```
$Self->True(
    $EventID,
    'EventAdd()',
);
```

False()

This function tests whether the return value of the function 'EventAdd()' in the variable \$EventID is invalid.

```
$Self->False(
    $EventID,
    'EventAdd()',
);
```

Is()

This function tests whether the variables \$A and \$B are equal.

```
$Self->Is(
    $A,
    $B,
    'Test Name',
);
```

Appendix A. Additional Resources

OTRS.org

The OTRS project website with source code, documentation and news is available at:

<http://otrs.org/>

Online API Library

The OTRS developer API documentation is available at:

<http://dev.otrs.org/>

Developer Mailing List

The OTRS developer mailing list is available at:

<http://lists.otrs.org/>

Commercial Support

For services (support, consulting, development, and training) you can contact the company behind OTRS, OTRS AG. They have offices in Germany, USA, Mexico, the Netherlands and other countries. Look at their website for contact information: <http://www.otrs.com/en/corporate-navigation/contact/>